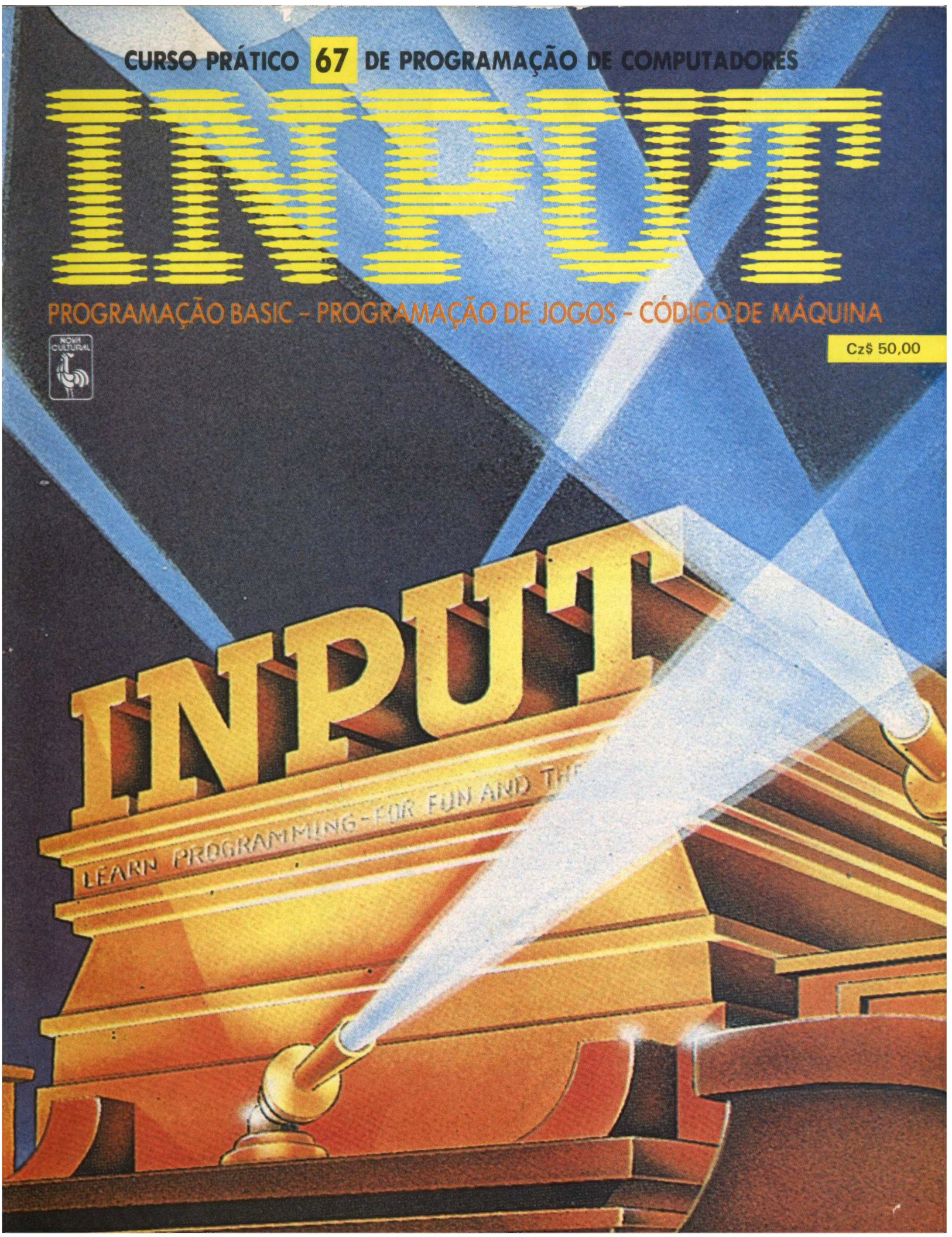
CURSO PRÁTICO 67 DE PROGRAMAÇÃO DE COMPUTADORES

INPUT

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

NOVA CULTURAL

Cz$ 50,00



INPUT

Vol. 5 Nº 67

NESTE NÚMERO

PERIFÉRICOS

CONTROLE POR COMPUTADOR

Sistemas de controle. Temporização e seqüenciamento. Regulação. Redução de dados. Sensores e atuadores. Conexão ao micro. Software 1321

LINGUAGENS

O LOGO E A TARTARUGA

Procedimentos com variáveis. Círculos e polígonos. Apagando o trabalho. Repetição e recursão. Interrupção. Armazenagem dos programas..... 1326

PROGRAMAÇÃO DE JOGOS

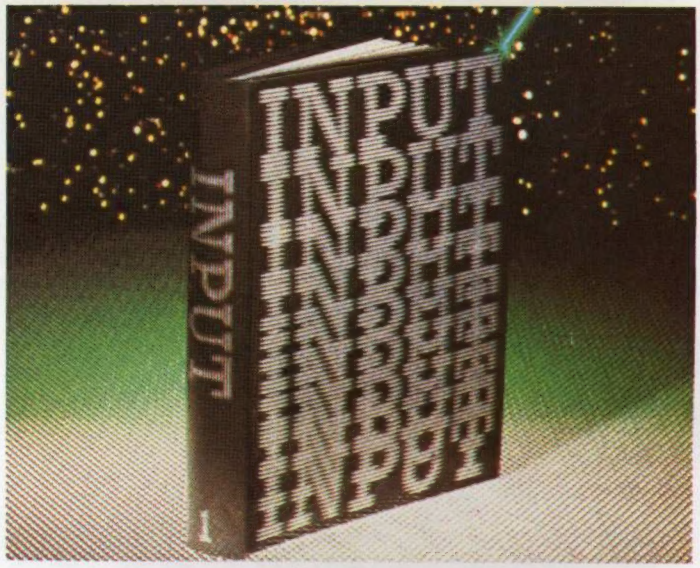
COMPRESSÃO DE TEXTOS (1)

Armazenagem do texto. Economia de espaço. Teoria da compressão de textos. Substitutos para o código ASCII. O uso da estatística..... 1332

PROGRAMAÇÃO BASIC

OS SEGREDOS DO SPECTRUM (2)

Variáveis do sistema. Modificação de apontadores e contadores. Endereços úteis..... 1340



PLANO DA OBRA

"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: 1. PESSOALMENTE — Por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em **São Paulo**, os endereços são: rua Brigadeiro Tobias, 773, Centro; avenida Industrial, 117, Santo André; e no **Rio de Janeiro**: avenida Mem de Sá, 191/193, Centro. 2. POR CARTA — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidora Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132, Jardim Teresa — CEP 06000 — Osasco — SP. Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na agência do Correio. 3. POR TELEX — Utilize o nº (011) 33 670 DNAP.

Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações, Lda. — Qta. Pau Varais, Azinhaga de Fetais — 2 685, Camarate — Lisboa; Apartado 57 — Telex 43 069 JARLIS P.

Atenção: Após seis meses do encerramento da coleção, os pedidos serão atendidos dependendo da disponibilidade do estoque.

Obs.: Quando pedir livros, mencione sempre título e/ou autor da obra, além do número da edição.

COLABORE CONOSCO

Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.

EDITOR
RICHARD CIVITA

NOVA CULTURAL

Presidente
Flávio Barros Pinto
Diretoria
Carmo Chagas, Iara Rodrigues, Pierluigi Bracco, Plácido Nicoleto, Roberto Silveira, Shozi Ikeda, Sônia Carvalho

REDAÇÃO
Diretor Editorial: Carmo Chagas
Editores Executivos:
Antonio José Filho, Berta Sztark Amar
Editor Chefe: Paulo de Almeida
Editoras Assistentes: Ana Lúcia B. de Lucena, Marisa Soares de Andrade
Chefe de Arte: Carlos Luiz Batista

Assistentes de Arte: Dagmar Bastos Sampaio, Grace Alonso Arruda, Monica Lenardon Corradi

Colaboradores
Consultor Editorial Responsável:
Dr. Renato M. E. Sabbatini
(Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)
Execução Editorial: DATAQUEST Assessoria em Informática Ltda., Campinas, SP
Tradução, adaptação, programação e redação:
Abílio Pedro Neto, Aluísio J. Dornellas de Barros, Marcelo R. Pires Therezo, Marcos Huascar Velasco, Raul Neder Porrelli, Ricardo J. P. de Aquino Pereira
Coordenação Geral: Rejane Felizatti Sabbatini

COMERCIAL
Diretor Comercial: Roberto Silveira
Gerente Comercial: Flávio Maculan
Gerente de Circulação: Denise Mozol
Gerente de Propaganda e Publicidade: José Carlos Madio
Gerente de Pesquisa e Análise de Mercado: Wagner M. P. Nabuco de Araújo

CLC

A Editora Nova Cultural Ltda. é uma empresa do Grupo CLC — Comunicações, Lazer e Cultura
Presidente: Richard Civita
Diretoria: Flávio Barros Pinto, João Gomez, Menahen M. Politi, Renê C. X. Santos, Stélio Alves Campos

© Marshall Cavendish Limited 1984/85.
© Editora Nova Cultural Ltda., São Paulo, Brasil, 1986.

Edição organizada pela Editora Nova Cultural Ltda.
Av. Brigadeiro Faria Lima, nº 2000 - 3º andar
CEP 01452 - São Paulo - SP - Brasil
(Artigo 15 da Lei 5 988, de 14/12/1973).
Esta obra foi composta na AM Produções Gráficas Ltda. e impressa na Divisão Gráfica da Editora Abril S.A.

CONTROLE POR COMPUTADOR

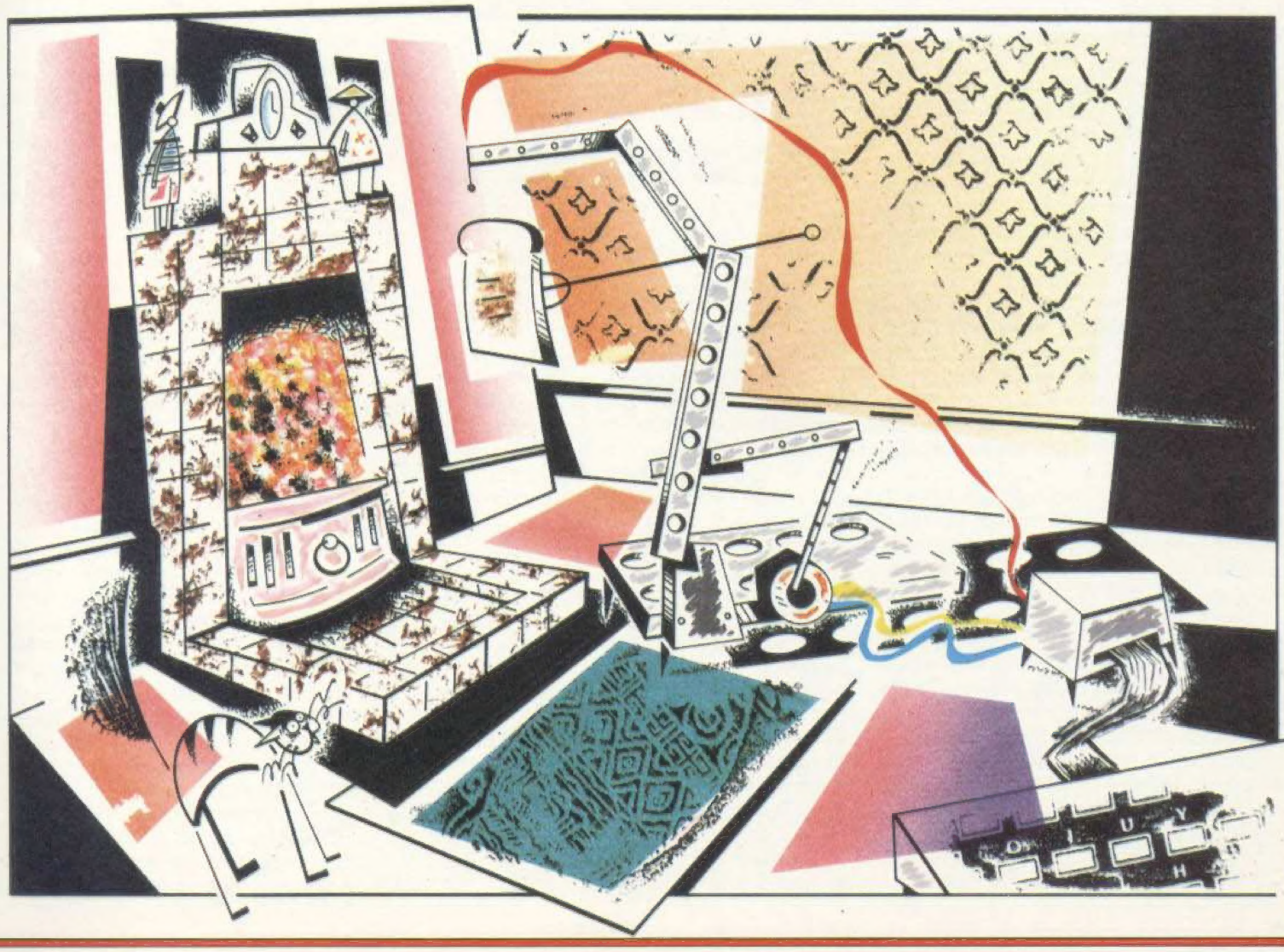
- CONTROLES PELA MICROELETRÔNICA
- SENSORES E ATUADORES
- CONEXÃO AO MICRO
- SOFTWARE

Como um computador pode receber informação do mundo exterior, analisá-la e usar os dados para operar dispositivos de diversos sistemas? Explicamos aqui todo o processo.

A microeletrônica provocou uma verdadeira revolução nos sistemas de controle de uma grande variedade de máquinas de uso doméstico e industrial. Dezenas de relés, cronômetros mecânicos e motores elétricos, usados antigamente para controlar aparelhos domésticos, foram substituídos por uma pequena caixa. Os circuitos e componentes microeletrônicos nela contidos realizam a mesma função de modo muito mais eficiente e sem falhas e avarias. Na indústria, inúmeros tipos de máquinas, como tornos e cortadores computadorizados e robôs de montagem, transformaram o dia-a-dia das fábricas.

Existem quatro tipos básicos de sistema microeletrônico de controle: circuitos lógicos discretos, circuitos integrados especiais, circuitos integrados genéricos e microprocessadores. Dentre todos, os microprocessadores foram os mais revolucionários, pois, além de ampliar o alcance das aplicações, eles barateram o custo dos sistemas de controle, através do conceito de *controle por software*. Um microprocessador tem uma infinidade de aplicações, bastando reprogramá-lo. Os outros três tipos de sistema, ao contrário, não podem ser mudados, a não ser que se façam modificações no hardware.

Diversos tipos de equipamentos e aparelhos contêm microprocessadores embutidos, que funcionam com base em softwares específicos para cada tarefa de controle. Evidentemente, apenas o microprocessador não é suficiente para essa aplicação: são necessários *chips* auxiliares adicionais (circuitos integrados), conectados ao processador, para que ele possa exercer alguma função útil. Assim, todo proprietário de um computador pessoal tem em suas mãos o coração de um sistema potencialmente ver-



sátil de controle. O usuário que quiser utilizar seu micro para esse fim encontrará uma grande variedade de interfaces especiais e *kits* de montagem. E, se for um amador curioso, poderá usar sua criatividade e testar uma série enorme de aplicações.

POSSIBILIDADES

Para que um microcomputador pessoal comande um sistema sofisticado de controle, é necessário acrescentar-lhe uma série de dispositivos.

Uma aplicação típica de controle sempre requer uma ou mais entradas e saídas especiais, operadas de maneira bem diferente das entradas e saídas mais comuns em um microcomputador. Um mecanismo de controle de temperatura, por exemplo, precisa ser ativado quando a temperatura atinge um determinado grau. O computador recolhe essa informação por intermédio de um *sensor*, que gera um sinal relativo ao fenômeno físico ou químico que está sendo detectado e o envia ao computador. Para agir sobre o mundo exterior no sentido de combater esse aumento de temperatura, o computador deverá estar ligado a um *atuador* — no caso, um condicionador de ar ou um ventilador.

No artigo sobre robôs (página 1284), descrevemos operações desse tipo. Os kits de braços robóticos, ali mencionados, são um exemplo de sistemas de controle especializados.

Neste artigo, examinamos os aspectos gerais do emprego de computadores pessoais no controle de dispositivos externos. Nossa discussão será centrada nos três principais elementos de um sistema de controle: sensores, atuadores e conectores. Antes, porém, analisaremos as possibilidades de controle.

Convém explorar mais detalhadamente quatro áreas:

- temporização e seqüenciamento
- regulação
- redução de dados
- interfaces homem-máquina

TEMPORIZAÇÃO E SEQÜENCIAMENTO

Sistemas de temporização e seqüenciamento fazem parte do nosso cotidiano: são usados nos programadores de máquinas de lavar, máquinas de costura, aquecimento central etc. O computador que administra muitos desses sistemas é bem mais compacto e confiável do que os sistemas eletromecânicos que substitui. Ele possibilita a execução de seqüências complexas de operação e cobre uma área muito maior de alternativas de aplicação. Além disso, permite que a temporização e seqüência de operações sejam alteradas, bastando, para isso, mudar o software.

Uma aplicação bastante simples do micro como o "cérebro" de um sistema de controle refere-se ao alarme residencial contra ladrões. Este consiste, basicamente, em uma série de fios conectados a interruptores localizados em várias portas e janelas da casa. Quando todas estão fechadas, o circuito se completa e a corrente elétrica pode fluir pelo mesmo. Se o circuito for interrompido pela abertura de uma janela ou porta, o alarme soa.

Um dos problemas com sistemas desse tipo é a possibilidade relativamente alta de um falso alarme, provocado por falhas mecânicas ou mau contato nos interruptores. O computador oferece uma opção eficiente para a solução desse problema. Se, por exemplo, acrescentarmos um segundo circuito a todas as portas internas, e conectarmos ambos os circuitos a um microcomputador, o software poderá realizar alguns testes simples, antes de soar o alarme. Isso seria feito segundo uma determinada lógica: se um ladrão entrar na casa, uma porta interna deverá ser aberta logo após a interrupção do circuito externo. O alarme será ativado se os dois circuitos forem interrompidos na ordem correta — ou seja, do exterior para o interior. Caso os circuitos independentes sejam interrompidos isoladamente, ou em ordem incorreta, o alarme não é ativado.

O uso do computador no controle do sistema antiladrões apresenta outras vantagens. Por exemplo: dispondo de um sistema adequado de telecomunicações, podemos fazer com que o micro disque automaticamente um número de telefone e avise a polícia.

Em algumas residências, um interruptor simples é utilizado para ligar e desligar luzes internas, para dar a impressão — na ausência dos moradores — de que há gente em casa. Mas o feitiço pode se virar contra o feiticeiro, pois a seqüência regular de liga-desliga evidencia exatamente o oposto: que não há

ninguém! Com um micro, é possível controlar tanto a iluminação dos aposentos como o funcionamento dos eletrodomésticos ligando-os segundo um padrão complexo, diferente para cada período do dia e variável de um dia para outro. Seqüências diversas podem ser deflagradas por eventos externos, como o nascer ou o pôr-do-sol.

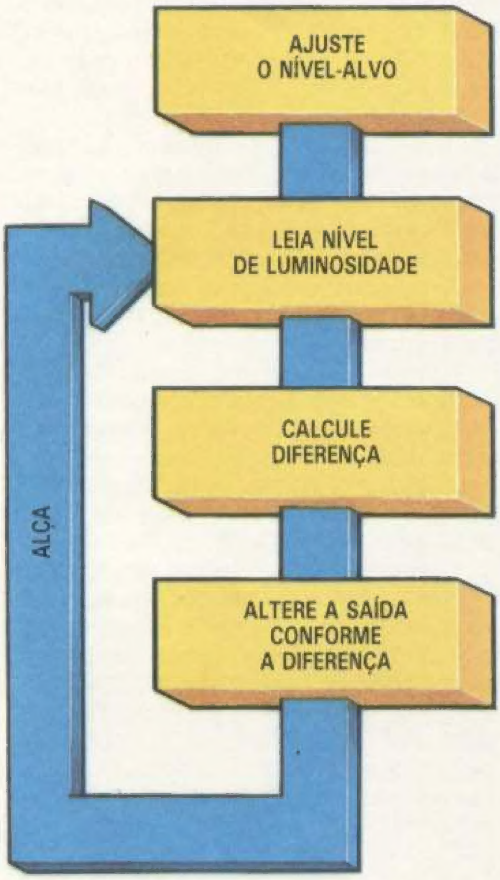
Outro candidato ideal para controle por micros é a ferrovia-modelo: um software adequado pode ser usado para verificar cruzamentos, acionar semáforos, desviar trilhos etc. Também interessante, no plano doméstico (ou até profissional), é o uso do micro para controlar um ou mais projetores de slides, de modo a criar um show mais complexo. O micro define a intensidade das lâmpadas — permitindo a criação de *fade-in* e *fade-out*, mixagens de imagens de dois projetores etc. —, a posição dos slides numa dada seqüência e informa qual imagem será projetada em cada máquina. O micro encarrega-se, ainda, da sincronização de música e fala, gravadas em uma fita cassete.

CONTROLES DE REGULAÇÃO

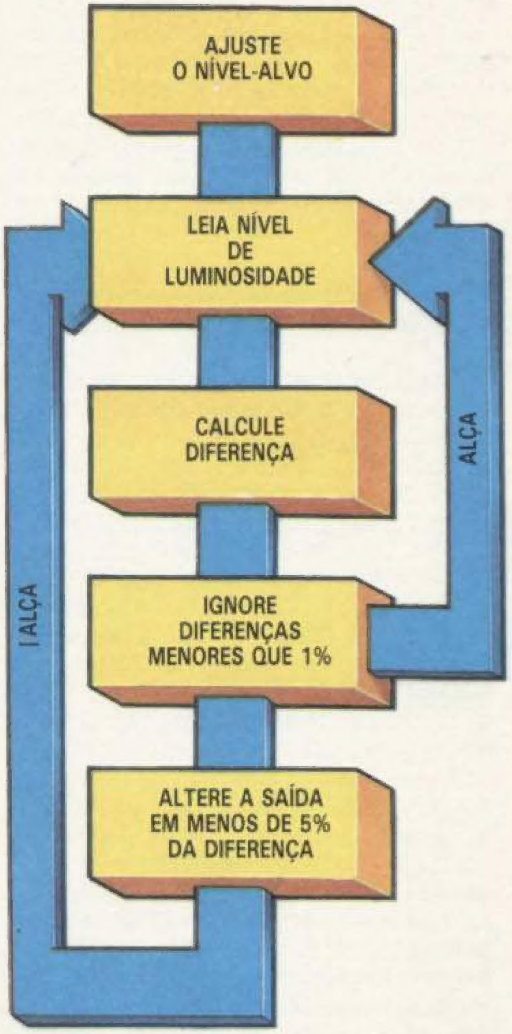
O controle de regulação é usado em toda aplicação em que a diferença entre uma situação ideal e a situação atual determina a operação a ser efetuada. Essa diferença é avaliada a partir de sensores conectados ao sistema. Um controlador de aquecimento central, por exemplo, tem a função de manter constante a temperatura do ambiente. Um sensor informa a temperatura do ambiente ao sistema de controle e este liga ou desliga o aquecedor conforme a temperatura caia ou suba.

Um microcomputador substitui facilmente sistemas desse tipo, permitindo operações mais complexas, como controle simultâneo de aposentos, do tanque de armazenamento de água quente etc. Para isso, ele aciona válvulas, bombas e elementos de aquecimento conforme a necessidade. Este sistema é um exemplo de um *servomecanismo* — baseado em controle por *alça fechada* (*feedback*, ou retroalimentação) — no qual a correção é proporcional ao erro detectado (*controle proporcional*).

Examinando mais detidamente o conceito de alça fechada, entenderemos por que o computador é tão versátil. Suponhamos que você queira regular a intensidade de uma lâmpada incandescente, de modo que o nível de luminosidade do ambiente seja constante. A variação da intensidade deve refletir as mudanças da luminosidade externa, ou seja, a lâmpada ficará mais fraca durante o dia, e mais forte à noite. Para isso, o computador é conectado a uma interface, que controla a corrente elétrica fornecida à lâmpada, e a um sensor fotelétrico, que mede o nível de luz no ambiente e fornece ao computador essa informação. Um sistema de retroalimentação de alça fechada é então incorporado, na forma de um software adequado, segundo este esquema:



Quando o programa for executado, a luminosidade da lâmpada começará a oscilar, tornando-se mais forte e, em seguida, mais fraca que o nível pretendido (nível-alvo). Esse efeito decorre de um atraso no sistema de controle, causado pela defasagem térmica da lâmpada — ou seja, ela não é capaz de responder à velocidade dos comandos do computador. Quando a corrente elétrica para a lâmpada é alterada, seu filamento demora a esquentar ou esfriar. Entretanto, o computador já determinou uma outra medida de luminosidade, e, comprovando que ainda existe um erro, ou diferença, aumenta o grau de correção. Eventualmente, a lâmpada "alcança" o computador e, se a correção efetuada é grande demais, chega a "ultrapassá-lo", invertendo o processo. Para evitar essa oscilação, algumas regras simples devem ser acrescentadas:



Agora, o sistema irá ajustar gradualmente a intensidade da luz e mantê-la constante, no nível-alvo. Outras regras podem ser incorporadas para a promoção de ações corretoras no caso de mudanças abruptas ou transitórias na iluminação externa.

REDUÇÃO DE DADOS

Aquisição e redução de dados e interfaces homem-máquina são elementos importantes de qualquer sistema de controle. Em muitas aplicações, o micro recebe informações de fontes diferentes, e tem que convertê-las para uma forma mais inteligível ou adequada. O software deve verificar se os níveis permanecem dentro de certos limites, converter unidades de medidas, filtrar os dados e checar inconsistências lógicas. Esse conjunto de atividades é chamado *redução de dados*.

Um analisador automotivo, para "acertar" a ignição de motores, constitui um bom exemplo de aplicação para as técnicas de redução de dados. Um sistema desse tipo exigirá sensores capazes

de monitorar o nível de dióxido de carbono na exaustão, o sincronismo da ignição, o ângulo de avanço do distribuidor etc. Ele poderia ser interativo — ou seja, o micro daria todas as instruções para a ligação dos sensores ao motor, acusaria eventuais falhas na operação e até mesmo diagnosticaria o problema do motor.

INTERFACES HOMEM-MÁQUINA

O que chamamos interface homem-máquina é exatamente o projeto da interação do sistema com o usuário. Esse elemento dos sistema de controle oferece enormes possibilidades de desenvolvimento. Mesmo ao nível mais simples, que consiste na exibição de determinada informação, há numerosas alternativas, como a simulação no vídeo de um mostrador analógico ou uma barra iluminada, em vez de complexas tabelas de números ou outros tipos de gráfico com os quais o usuário está menos habituado. As opções incluem a representação de um esquema do sistema que está sendo controlado. No caso de uma ferrovia de brinquedo, por exemplo, é possível traçar no vídeo os trilhos e os pontos de cruzamento, e assinalar em tempo real o seu "status". O usuário pode comandar todas as funções — como selecionar trens, mudar sua velocidade etc. — por meio de joysticks ou canetas ópticas.

O micro também abre novas perspectivas para o projeto e controle de sistemas de auxílio para deficientes físicos. Esses sistemas permitem, por exemplo, que um paralítico controle seu ambiente por meio de um computador, ligando ou desligando eletrodomésticos, alterando o nível da calefação ou a iluminação de um aposento etc.

Já existem diversas interfaces homem-máquina especiais para deficientes, como as de entrada e saída baseadas na voz ou aquelas em que se acionam as teclas soprando-se em um tubinho. Com isso, uma pessoa deficiente pode até mesmo usar um processador de textos, sem necessidade do teclado.

SENSORES

As formas de controle por meio de computadores que acabamos de examinar são bastante inter-relacionadas, e apresentam uma série de elementos em comum. Um dos mais importantes é a maneira como os computadores obtêm informações do mundo exterior.

Sensores especializados, correspondentes aos cinco sentidos — tato, olfato, visão, audição e paladar —, podem ser ligados a um microcomputador. Eles não chegam a ter o nível de complexidade dos sentidos humanos. Em compensação, os micros contam, potencialmente, com "sentidos" adicionais, como a capacidade de detectar campos magnéticos, radiação atômica, eletromagnética, de ultra-som ou raios X, infravermelha ou ultravioleta etc.

O computador precisa de um sensor específico para cada tipo de fenômeno a ser detectado. Existem, assim, detectores de som e ultra-som, temperatura, gases, fluxo, umidade, luz, proximidade, movimento, aceleração etc.

Seja qual for o tipo de energia à qual o sensor é sensível, geralmente a forma de saída é de natureza elétrica — como uma corrente ou uma voltagem analógica ou digital. Dá-se o nome de *transdutor* ao sensor que se encarrega de realizar a conversão de um tipo de energia para outro.

Como sabemos, o computador não pode trabalhar diretamente com sinais analógicos: é preciso, antes, convertê-los para sinais digitais, ou números binários. Isso é feito por uma interface especial de conversão, chamada *conversor analógico-digital* (AD).

Alguns sensores dispõem de algum tipo de circuito de pré-processamento, embutido — como amplificadores de sinal, condicionadores, filtros ou conversores AD simples — o que facilita o trabalho de conexão ao computador. Os detectores de temperatura (termistores), por exemplo, não respondem de forma inteiramente linear à variação da temperatura. Alguns termistores "inteligentes", porém, já apresentam o sinal de saída corrigido conforme o nível de temperatura, dispensando correções por parte do software de controle.

ATUADORES

Até agora, vimos como o computador "sente" o que está acontecendo no ambiente. Entretanto, para agir sobre o mesmo, ele precisa de atuadores, elementos que geram alguma forma de energia: luminosa, elétrica, mecânica, magnética etc. Na maioria das aplicações, os atuadores são eletromecânicos, dividindo-se, basicamente, em cinco tipos:

- pneumáticos
- hidráulicos
- motores elétricos DC ou AC
- solenóides
- motores de passo

Os atuadores pneumáticos, bem como os hidráulicos, utilizam a pressão gerada por fluidos (gases ou líquidos) e compõem-se de um cilindro contendo um pistão, conectado a um braço de impulsão ou de tração. Uma válvula operada eletricamnte dirige o fluxo de líquido ou gás para o pistão. Os atuadores hidráulicos geralmente são mais seguros, potentes e precisos do que os pneumáticos.

Motores comuns de corrente direta (DC) ou alternada (AC) podem ser controlados por um micro, embora a precisão seja pequena. Quando é necessário um posicionamento exato, deve-se utilizar um sensor do ângulo de rotação do eixo, que informa ao computador a posição e velocidade. Esses codificadores, como são chamados, transformam o ângulo de rotação em um número de bits. Podem ser ópticos ou elétricos — nesse último caso, temos um conjunto *servomotor*.

Devido às complicações mencionadas e ao alto preço de um servomotor, os sistemas baseados em micros usam com maior freqüência os solenóides ou os motores de passo. Mais simples — já que precisam do computador só para ligá-los e desligá-los, não exigindo controle proporcional — esses atuadores adaptam-se melhor ao mundo digital.

Um solenóide é um eletroímã que aciona algum tipo de eixo (ou armadura). Quando a corrente elétrica passa através de uma bobina, o campo magnético resultante move a armadura. O relé é um solenóide, só que destinado a ligar e desligar correntes elétricas.

O motor de passo (*stepper*) utiliza uma armadura circular de solenóides para girar um eixo, do mesmo modo que um motor elétrico comum. A diferença é que o avanço se faz em pequenos passos, e não em movimentos contínuos. A utilização do stepper em conjunto com um micro é simples, mas esse atuador apresenta uma desvantagem: sua potência é bem inferior à de motores elétricos de tamanho semelhante.

Para entender o funcionamento de um motor de passo, tomemos como exemplo um modelo simplificado com apenas quatro pólos, ou seja, quatro eletroímãs dispostos em ângulos retos ao redor do rotor. O rotor é uma armadura de ferro, com duas peças polares, e gira livremente sobre um eixo. Se um dos magnetos é acionado, o rotor é atraído para ele, e gira um certo número de graus. Caso um microcomputador acione em seqüência cada um dos magnetos, o rotor girará um número preciso de vezes — efeito impossível de se obter com um motor linear. Quanto mais

rápida a atuação do micro, mais depressa irá girar o rotor. Os motores de passo têm um grande número de bobinas (cerca de duzentas, em média), e, conseqüentemente conseguem uma rotação muito mais "macia". O rotor pode ser posicionado sem necessidade de servomecanismos que indiquem sua posição.

CONEXÃO AO COMPUTADOR

A maioria dos sensores e atuadores exige ou fornece uma voltagem entre cinco e dez volts, muitas vezes com uma corrente de valor elevado. Como o micro não foi projetado para trabalhar diretamente com essas voltagens e correntes, a conexão às portas de entrada e saída exige uma interface.

As interfaces disponíveis oferecem saídas comutadas de corrente ou voltagem de valor elevado, entradas por interruptores, portas binárias de oito a dezesseis bits, conversores AD ou DA de média ou alta velocidade etc.

Muitos micros possuem conversores AD embutidos (a porta de gravador cassete, por exemplo, ou a entrada de joysticks), mas estes não têm freqüências de amostragem (número de conversões feitas por segundo) tão altas quanto as requeridas por muitas aplicações. Por isso, conversores extras são necessários. Um conversor DA, por sua vez, fornece voltagens ou correntes contínuas, a partir de sinais binários, para acionar motores DC, lâmpadas, alto-falantes e assim por diante.

Alguns sistemas de controle doméstico, como os destinados a ligar e desligar eletrodomésticos a distância, utilizam a própria rede de força da casa para enviar os sinais binários.

SISTEMAS MECÂNICOS

Existem várias alterativas para a ligação do computador a um atuador mecânico. Muitas delas foram usadas inclusive no desenvolvimento de brinquedos ou kits acionados por computadores. No exterior, encontram-se diversos sistemas — como o Meccano, o Lego e o Fischer — que são verdadeiros projetos de engenharia em miniatura, com todos os componentes imagináveis.

Quanto maior a flexibilidade do kit, melhor, pois conexões mecânicas quase sempre precisam ser projetadas de acordo com a tarefa. Um sistema de engrenagens, polias e alavancas, conectadas a relés, solenóides e motores, requer, é evidente, um minucioso trabalho de planejamento e construção.

SOFTWARE

Desenvolver software para aplicações de controle não é mais difícil do que para outros tipos de aplicação, como jogos, por exemplo. As interfaces disponíveis simplificam bastante esse trabalho. Muitas vezes, elas são fornecidas com uma biblioteca de rotinas em linguagem de máquina ou BASIC, que podem ser utilizadas como módulos de um sistema mais complexo de controle, escrito pelo usuário.

Um dos passos mais importantes no desenvolvimento do software consiste na análise e divisão do problema de controle em partes menores, de tal forma que se possa escrever e testar individualmente cada seção, antes de integrá-la ao sistema maior.

A necessidade de manter o microcomputador permanentemente conectado ao sistema de controle costuma desestimular o usuário a se aventurar nessa área. Esse inconveniente, porém, pode ser superado com a aquisição de um *sistema de desenvolvimento*, que é um micro mais simples, geralmente em uma única placa, específica para as tarefas de controle. Essa solução, além de ser barata, é a mais adequada: dificilmente uma aplicação de controle requer todos os recursos disponíveis em um computador de uso geral.

Os sistemas de desenvolvimento são fornecidos com um conjunto de hardware e software que permite a adaptação do computador à tarefa específica que será realizada. Se os dados a serem exibidos são exclusivamente numéricos, basta um visor LED ou de cristal líquido, do tipo existente para calculadoras. Caso a aplicação exija apenas algumas teclas ou botões, não é preciso acrescentar um teclado completo. Entretanto, sistemas desse tipo têm necessidades que um micro comum nem sempre é capaz de atender — como um número grande de canais analógicos, ou saída de controle de correntes altas.

O usuário pode modificar à vontade o software que acompanha os sistemas de desenvolvimento, incorporando-o, posteriormente, a uma memória EPROM, para uso permanente.

O LOGO E A TARTARUGA

Estrelas, círculos, espirais: como num passe de mágica, as mais variadas figuras surgem no vídeo — por obra e graça de uma divertida tartaruga. Comande você mesmo o espetáculo.

Já vimos como fazer desenhos com os comandos gráficos do LOGO usando a tartaruga — uma espécie de cursor triangular que indica a posição atual na tela. Podemos também "ensinar" a tartaruga a traçar determinadas formas através de um comando especial. Se quisermos, por exemplo, fazê-la desenhar um hexágono, digitamos:

```
TO HEXAGONO
REPEAT 6 [FORWARD 50 RIGHT 60]
END
```

```
AP HEXAGONO
REPITA 6 [PARAFRENTE 50
PARADIREITA 60]
FIM
```

Após digitarmos **END**, o interpretador indicará que o procedimento **HEXAGONO** foi definido, passando a fazer parte do vocabulário LOGO.

Quando você digitar a palavra **HEXAGONO**, a tartaruga desenhará um polígono regular de seis lados (cada lado com cinqüenta passos de comprimento), a partir da posição em que estiver. Depois, ela voltará ao ponto inicial, pois percorreu seis vezes 60 graus, ou seja, 360 graus. Esse "comportamento" sempre se repete com polígonos regulares, tendo recebido um nome jocoso dos entusiastas do LOGO: Teorema do Trajeto Total da Tartaruga (TTTT).

Um procedimento pode ser utilizado dentro de outros, como mostra o programa abaixo, chamado **FLOR**, que desenha doze hexágonos ao redor de um centro, formando, assim, as pétalas:

```
TO FLOR
REPEAT 12 [HEXAGONO FD 10 RT
30]
END
```

```
AP FLOR
REPITA 12 [HEXAGONO PF 10 PD
30]
FIM
```

Observe que deslocamos a tartaruga dez passos adiante, antes de repetirmos o traçado do hexágono — isto é, desviamos sua direção para 30 graus à direita da direção anterior.

A VISÃO DA TARTARUGA

Para fazer desenhos no vídeo, normalmente o BASIC usa como referencial as coordenadas de um sistema cartesiano (de eixos ortogonais, X e Y), com o qual estamos mais familiarizados. A geometria da tartaruga é diferente: nela, o ponto de referência é a própria tartaruga, ou seja, todos os deslocamentos lineares e angulares são realizados a partir da última posição em que a tartaruga se encontrava.

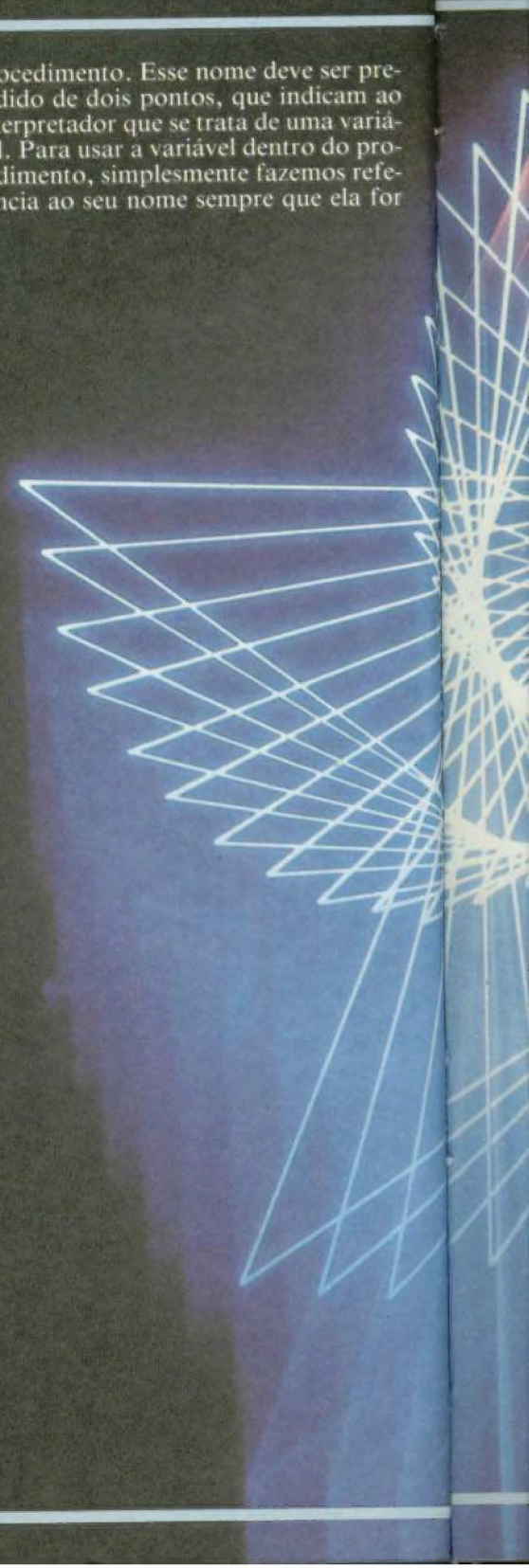
É por isso que usamos 60 graus para desenhar um hexágono, e não 120 graus (que corresponde ao ângulo entre os seus lados) — ou seja, temos que *virar* a tartaruga 60 graus para produzir um ângulo de 120 graus entre a última reta traçada e a próxima.

Parece confuso? Não se assuste: é bem mais fácil aprender geometria de uma forma intuitiva como esta, que nos permite "experimentar" o referencial da tartaruga, bastando imaginar que estamos em uma sala vazia, traçando alguma figura com nossos passos.

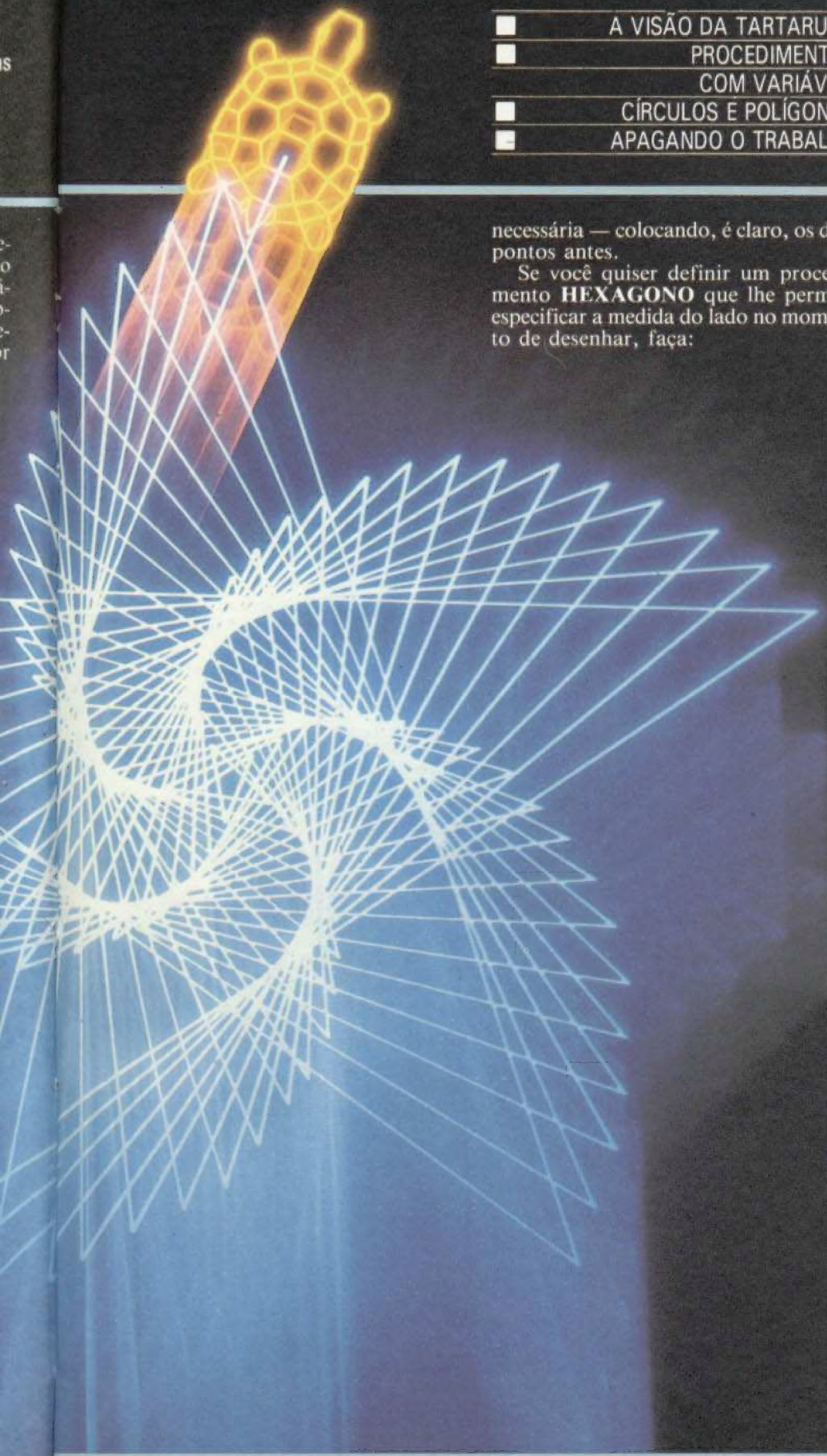
PROCEDIMENTOS COM VARIÁVEIS

Embora o procedimento **HEXAGONO** tenha sido incorporado ao vocabulário do LOGO, há uma importante diferença entre ele e vários dos comandos primitivos da linguagem, como aqueles que fazem a tartaruga avançar, recuar ou virar: o **HEXAGONO** traçará uma figura sempre do mesmo tamanho, ao passo que os comandos mencionados admitem um parâmetro variável.

Entretanto, é perfeitamente possível definir procedimentos que tenham parâmetros variáveis. Para isso, precisamos atribuir um nome ao parâmetro de entrada e incluí-lo na linha de título do procedimento. Esse nome deve ser precedido de dois pontos, que indicam ao interpretador que se trata de uma variável. Para usar a variável dentro do procedimento, simplesmente fazemos referência ao seu nome sempre que ela for



- A VISÃO DA TARTARUGA
- PROCEDIMENTOS COM VARIÁVEIS
- CÍRCULOS E POLÍGONOS
- APAGANDO O TRABALHO
- REPETIÇÃO E RECURSÃO
- COMO PARAR UMA REPETIÇÃO
- ARMAZENAGEM DOS PROGRAMAS



necessária — colocando, é claro, os dois pontos antes.

Se você quiser definir um procedimento **HEXAGONO** que lhe permita especificar a medida do lado no momento de desenhar, faça:

```
TO HEXAGONO :LADO
REPEAT 6 [FORWARD :LADO RIGHT
60]
END
```

```
AP HEXAGONO :LADO
REPITA 6 [PARAFRENTE :LADO
PARADIREITA  60]
FIM
```

Agora, para desenhar um hexágono, bastará digitar a palavra do procedimento, seguida do número correspondente ao comprimento desejado para o lado. Se você se esquecer de colocar esse número, o interpretador LOGO imprimirá uma mensagem de erro.

Digitando **HEXAGONO 20**, por exemplo, você fará a tartaruga desenhar um hexágono com vinte unidades (ou passos) de lado; com **HEXAGONO 40**, obterá um outro hexágono, com quarenta unidades de lado. O uso de **:LADO** dentro da linha de repetição do procedimento provocará a substituição automática do valor numérico anterior pelo especificado na entrada. O novo comando, **HEXAGONO**, passa assim a funcionar como os primitivos do LOGO.

Façamos o mesmo com **FLOR**, reescrevendo o procedimento:

```
TO FLOR :LADO
REPEAT 12 [HEXAGONO :LADO FD 10
RT 30]
END
```

```
AP FLOR :LADO
REPITA 12 [HEXAGONO :LADO PF 10
PD 30]
FIM
```

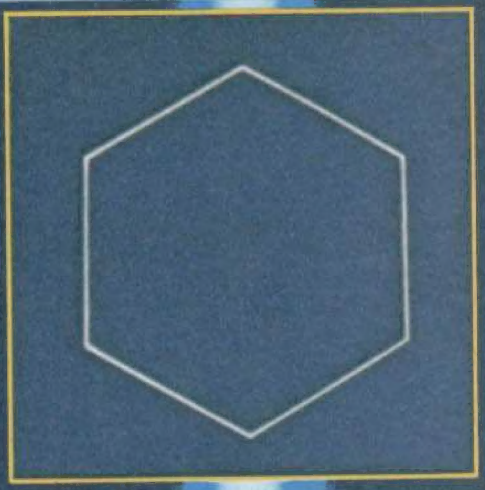
Quando digitarmos novamente **FLOR**, deveremos fornecer o tamanho desejado. Experimente várias medidas, para ver o resultado. Da mesma maneira, podemos tornar variáveis o avanço e o ângulo executados a cada novo hexágono:

```
TO FLOR :LADO :AVANCO :ANGULO
REPEAT 12 [HEXAGONO :LADO FD
:AVANCO RT :ANGULO]
END
```

```
AP FLOR :LADO :AVANCO :ANGULO
REPITA 12 [HEXAGONO :LADO PF
:AVANCO PD :ANGULO]
FIM
```

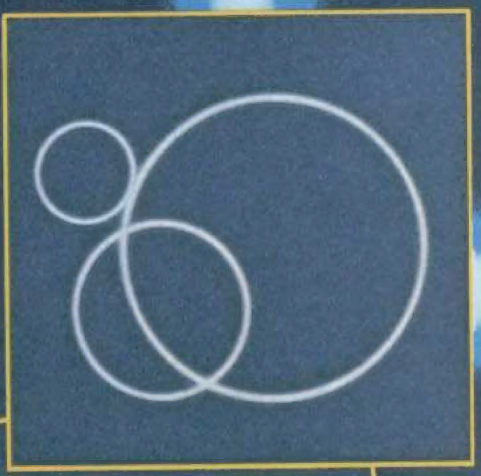
Com a alteração sistemática dos três parâmetros, é possível produzir desenhos muito diferentes entre si. Tente! Esta é a filosofia LOGO: estimular a experimentação e a criatividade, empregando programas extremamente simples, elegantes e, sobretudo, fáceis de entender. Qual seria o tamanho de um programa equivalente em BASIC, para fazer a mesmíssima coisa?

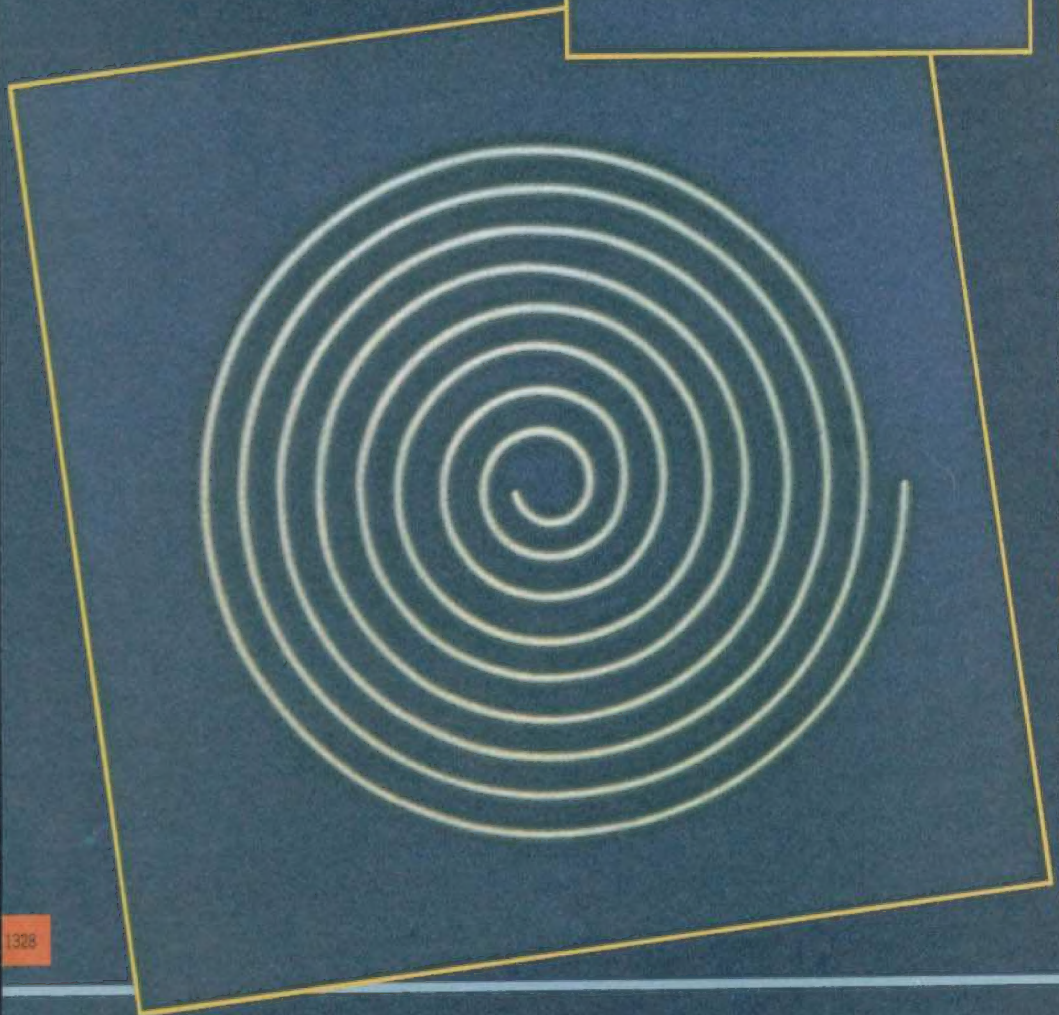
O Teorema do Trajeto Total da Tartaruga assegura que basta dividir 360 pelo número de lados do polígono regular, para se obter o ângulo de virada entre cada lado do mesmo. Poderíamos, portanto, definir um programa geral, que traça triângulos, quadrados, pentágonos, hexágonos etc., desde que especificássemos o número de lados e o comprimento de cada lado:



```
TO POLIGONO :NUMEROLADOS :LADO
REPEAT :NUMEROLADOS [FORWARD
:LADO RIGHT 360/:NUMEROLADOS]
END
```

```
AP POLIGONO :NUMEROLADOS :LADO
REPITA :NUMEROLADOS [PF :LADO
PD 360/:NUMEROLADOS]
FIM
```





Como mostra esse programa, o LOGO é capaz de resolver expressões matemáticas, à semelhança do BASIC. Se digitarmos **POLIGONO 8 20**, por exemplo, a substituição será feita automaticamente, e teremos a execução do seguinte comando (que traçará um octógono, com vinte unidades de lado):

```
REPEAT 8 [FORWARD 20 RIGHT 45]
```

```
REPITA 8 [PF 20 PD 45]
```

O mesmo procedimento pode ser utilizado, aliás, no traçado de uma circunferência. Para isso, precisamos aumentar bastante o número de lados — por exemplo, **POLIGONO 2 72**. Como os lados são demasiado pequenos, teremos a impressão de que a figura resultante é formada por uma linha contínua.

A tartaruga leva longo tempo para

traçar essa figura, pois seus deslocamentos são muito numerosos e, em cada ponto, seu símbolo tem que ser novamente desenhado. Esse inconveniente pode ser evitado por meio de um comando que torna a tartaruga invisível:

```
HIDETURTLE
```

```
DESAPAREÇATAT
```

O comando do MSX também pode ser abreviado por DT. Para fazer reaparecer a tartaruga, digitamos:

```
SHOWTURTLE
```

```
APAREÇATAT
```

Poderíamos, portanto, definir um procedimento específico para traçar círculos, incorporando todo o conhecimento que adquirimos até agora:

```
TO CIRCULO :LADO
HIDETURTLE
REPEAT 72 [FD :LADO RIGHT 5]
SHOWTURTLE
END
```

```
AP CIRCULO
DESAPAREÇATAT
```

```
REPITA 100 [PARADIREITA 36
PARAFRENTE 1]
APAREÇATAT
FIM
```

MODIFICAÇÃO E ARMAZENAGEM

À medida que vamos incorporando mais e mais procedimentos ao vocabulário corrente da linguagem LOGO, duas providências se tornam necessárias: modificar um procedimento já existente (para não ser preciso digitar todos os seus comandos de novo, a cada pequena alteração que queiramos fazer), e armazenar todos os procedimentos em disco ou fita cassete, para uma futura execução ou modificação.

O LOGO dispõe de um editor simples, que é chamado por intermédio do comando **EDIT** (**EDITE**, para as versões em língua portuguesa). Essa palavra-chave deve ser seguida do nome do procedimento a ser editado.

O comando **EDIT** apaga a tela e lista o procedimento pedido no vídeo, a partir do topo. Digitando uma série de teclas especiais (que variam conforme a versão do LOGO utilizada), podemos inserir, apagar e escrever caracteres e linhas, modificando o procedimento. Ao terminar a edição, pressionamos <**CTRL**> <**C**>, <**CTRL**> <**BREAK**> ou qualquer outra combinação de teclas específica do programa editor. Com isso, voltamos ao modo direto do LOGO.

Para armazenar o conjunto de procedimentos já definidos na memória, usamos o seguinte comando:

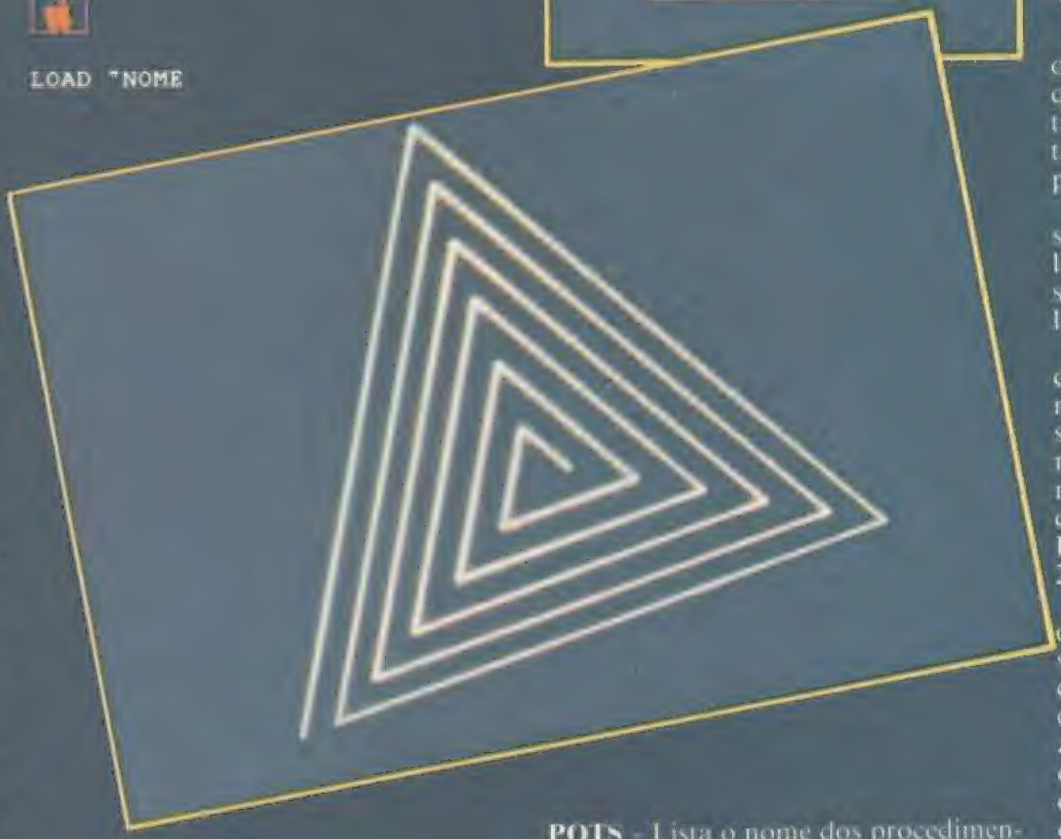
```
SAVE " NOME
```

```
GRAVETUDO "NOME
```

Nesse exemplo, o arquivo criado em fita, contendo todos os procedimentos, chama-se **NOME**. Essa palavra deve ser precedida por aspas (não é necessário fechá-las) — sinal convencional do LOGO para definir um literal.

Os procedimentos podem ser recuperados do disco ou fita pelo comando:

```
LOAD "NOME
```



```
CARREGUE "NOME
```

Ao usarmos o comando de carregamento, os novos procedimentos serão acrescentados aos que já se encontram na memória. Caso haja algum procedimento com o mesmo nome, ele desaparece, dando lugar ao procedimento carregado.

Existem também comandos para listar e apagar procedimentos nas memórias principal e auxiliar:

PO - Lista um procedimento nomeado (abreviatura de **PRINTOUT**.)

POTS - Lista o nome dos procedimentos existentes na memória.

CATALOG - Lista os arquivos existentes em disco.

ERASE - Apaga o procedimento nomeado (precedido de aspas) da memória principal.

ERASEFILE - Apaga o arquivo nomeado.

MO - Lista um procedimento nomeado (abreviatura de **MOSTRE**).

MOTS - Lista o nome dos procedimentos existentes em memória.

ARQUIVOS - Lista os arquivos existentes em disco.

ELIMINE - Apaga o procedimento nomeado (precedido de aspas) da memória principal.

ELIMINEARQ - Apaga o arquivo nomeado.

A forma desses comandos pode variar ligeiramente, segundo a versão do LOGO que está sendo utilizada. Consulte o manual de operação, antes de usá-los.

ORGANIZAÇÃO DA MEMÓRIA

A memória de trabalho do LOGO é organizada sob a forma de listas hierárquicas, incluindo no mesmo espaço tanto os nomes de variáveis e procedimentos, quanto os dos próprios comandos primitivos da linguagem.

Uma lista hierárquica pode ser representada graficamente por algo semelhante ao organograma de uma empresa ou, melhor ainda, pela árvore genealógica de uma família.

Cada nome definido no vocabulário do LOGO ocuparia um nodo — que corresponde a um ponto de bifurcação dessa árvore. Assim, a capacidade da memória, em LOGO, não é medida em bytes, mas em nodos. Dependendo do modelo do computador e também da versão do LOGO utilizada, essa medida varia de 200/300 nodos a mais de 30.000 nodos.

Sempre que se define um novo nome, o interpretador LOGO usa um nodo de seu espaço original. Quando um procedimento ou nome é apagado, uma série de nodos são liberados, passando a fazer parte de uma lista de nodos livres, que é então consultada pelo interpretador. Periodicamente, este realiza uma operação chamada "coleta de lixo", que consiste em reorganizar a estrutura da memória. Isto pode causar uma parada do LOGO durante alguns segundos, sem qualquer aviso prévio.

REPETIÇÃO E RECURSÃO

Como foi explicado anteriormente, a linguagem LOGO dispõe de um procedimento de repetição que, embora simples, requer a especificação do número de vezes que um determinado conjunto de comandos será repetido.

Com a utilização de um procedimento recursivo, torna-se bem mais fácil obter uma repetição.

Entende-se por recursão a capacidade que um procedimento tem de chamar a si mesmo. Algumas linguagens, como o LISP, o PASCAL, o LOGO, o C e o ALGOL, dispõem de capacidade recur-

siva infinita — ou seja, não há limite para o número de vezes que um procedimento pode chamar a si mesmo. Outras linguagens, como o BASIC e o FORTRAN, não possuem essa capacidade.

A recursão é uma das propriedades mais interessantes do LOGO, conferindo-lhe enorme poder computacional.

Veja, por exemplo, de que maneira poderíamos reprogramar o procedimento **FLOR**, utilizando a recursão:

```
TO FLOR :LADO
HEXAGONO FD :LADO RT 30
FLOR :LADO
END
```

```
AP FLOR :LADO
HEXAGONO PF :LADO PD 30
FLOR :LADO
FIM
```

O programa ficará preso, como se pode notar, em um laço infinito, que se repete indefinidamente.

Para interromper o programa, devemos pressionar <**BREAK**> ou <**CTRL**> <**G**>, dependendo do computador.

É possível incluir em um procedimento recursivo o número de parâmetros de entrada que quisermos. Um polígono regular, por exemplo, também pode ser traçado por um procedimento recursivo, que chamaremos **POLI**:

```
TO POLI :LADO :ANGULO
FORWARD :LADO RIGHT :ANGULO
POLI :LADO :ANGULO
END
```

```
AP POLI :LADO :ANGULO
PARAFRENTE :LADO PARADIREITA
:ANGULO
POLI :LADO :ANGULO
FIM
```

Com base no procedimento **POLI**, que, em si, nada faz de novo, podemos imaginar um outro, **ESPIRAL**, que traça desenhos variados, conforme os parâmetros com que é alimentado:

```
TO ESPIRAL :LADO :ANGULO
FORWARD :LADO RIGHT :ANGULO
ESPIRAL :LADO+2 :ANGULO
END
```

```
AP ESPIRAL :LADO :ANGULO
PARAFRENTE :LADO PARADIREITA
:ANGULO
ESPIRAL :LADO+2 :ANGULO
FIM
```

Observe que o LOGO acrescenta duas unidades ao **LADO**, em cada repetição do procedimento. Com isso, obtemos uma bonita espiral.

Tente os seguintes valores:

```
ESPIRAL 1 45
ESPIRAL 1 65
ESPIRAL 1 72
ESPIRAL 1 91
```

Se desejar que o desenho seja executado com maior rapidez, não se esqueça de tornar a tartaruga invisível.

COMO PARAR

Não existe nenhuma maneira de parar a tartaruga em recursão infinita, a não ser interrompendo o programa externamente. Isto não só é deselegante, como também impede que um procedimento recursivo seja chamado posteriormente por outros procedimentos, pois ele ficará imobilizado no laço infinito.

O LOGO oferece, porém, a possibilidade de se testar determinadas condições, por meio de uma operação bastante semelhante àquela que é efetuada pelo comando **IF...THEN** da linguagem BASIC. Podemos testar, por exemplo, se o lado da flor ou da espiral ultrapassou um certo limite, impondo, caso isto seja necessário, a parada e retorno do procedimento ao nível imediatamente superior de chamada (que corresponde a um outro procedimento ou ao modo direto). Para isso, utiliza-se um comando primitivo — **STOP** ou, nas versões em português, **PARE** — que não pode ser usado em modo direto.

Para esclarecer melhor o emprego desse primitivo, vamos programar um procedimento denominado **GIRAHEX**, que traça hexágonos com lados progressivamente maiores, até que o lado de tamanho 60 seja atingido. Recorreremos ao procedimento **HEXAGONO**, definido anteriormente, para desenhar a figura.

```
TO GIRAHEX :LADO :ANGULO
IF :LADO>60 THEN STOP
HEXAGONO :LADO
RIGHT :ANGLE
GIRAHEX :LADO+3 :ANGULO
END
```

MICRO DICAS

TRADUÇÃO PARA O MLOGO

Os comandos do MLOGO, para o Apple, correspondentes aos comandos do BRASLOGO usados neste artigo, são:

BRASLOGO	MLOGO
SEMTARTARUGA	SEMT
EDITE	EDITE
SE	SE
ENTAO	ENTAO
PARE	PARE
GRAVETUDO	GRAVE
CARREGUE	LEIA
MOPS	LISTAR
MO	LISTE
ELIMINE	—
ELIMINEARQ	SEMARQUIVO
ARQUIVOS	VERDISCO

```
AP GIRAHEX :LADO :ANGULO
SE :LADO>60 ENTAO [PARE]
HEXAGONO :LADO
PARADIREITA :ANGULO
GIRAHEX :LADO+3 :ANGULO
FIM
```

Na versão em BRASLOGO, o comando **PARE** está especificado como se fosse uma lista. Mais poderoso que o LOGO padrão, o BRASLOGO permite especificar, como mostramos a seguir, uma sequência de comandos que será executada se a condição **SE** for verdadeira.

```
SE :LADO>60 ENTAO [PARACENTRO
PARE]
```

O procedimento **GIRAHEX 1 10** faz o computador traçar um hexágono com uma unidade de lado, girar a tartaruga 10 graus, traçar um hexágono com quatro unidades de lado, girar novamente 10 graus e assim por diante. Quando o lado do hexágono tiver mais de cem unidades, o procedimento será interrompido automaticamente.

Com as informações contidas neste artigo, você já pode explorar à vontade as fabulosas propriedades da recursão, obtendo uma grande variedade de efeitos a partir de um mesmo programa. O que aconteceria, por exemplo, se também variássemos o **:ANGULO** a cada recursão, no programa **GIRAHEX**?

No terceiro e último artigo desta série de artigos, vamos investigar os recursos do LOGO no processamento de listas, palavras e equações matemáticas.

COMPRESSÃO DE TEXTOS (1)

As limitações da memória do micro costumam frustrar os programadores que apreciam jogos de aventura. Veja como fazer para colocar o máximo de texto em um mínimo de memória.

Todo programador enfrenta o mesmo problema ao desenvolver jogos de aventura: o texto não cabe no espaço disponível na memória do micro. Quem não pode comprar uma máquina maior ou expandir a memória da sua, costuma contornar essa dificuldade reduzindo o tamanho do jogo ou simplificando demais o texto. Muitas vezes, o programador se vê obrigado, por exemplo, a eliminar as instruções do programa — o que é mau para quem vai utilizá-lo.

Existe uma maneira, porém, de tornar o programa menos extenso. Os métodos normalmente empregados para encurtar um programa, discutidos no artigo da página 141, não são eficientes no que diz respeito a jogos de aventura, pois a maior parte destes consiste em textos (listas de objetos, locações, situações, mensagens, instruções etc.). A solução está, portanto, em reduzir o espaço ocupado pelo texto.

Examinaremos aqui várias técnicas, de compressão de texto e, em artigo posterior, veremos como usar uma delas em um jogo de aventuras.

COMO UM TEXTO É ARMAZENADO

O computador normalmente armazena o texto — ou seja, tudo o que não é instrução de programa, ou constante numérica — da mesma maneira em que foi entrado, usando o código ASCII (veja o artigo da página 361).

Recordando, o código ASCII (*American Standard Code for Information Interchange*, Código Padrão Americano para Intercâmbio de Informação) é um padrão utilizado para representar caracteres na memória do computador. Ele comporta 256 códigos ou caracteres diferentes. Os códigos 0 a 31 são reservados para funções de controle e os códigos 32 a 90, para caracteres alfabéticos, numéricos e de pontuação. Os códigos 91 e seguintes estão livres para outros usos, como caracteres gráficos, sinais de acentuação.

Duas características do ASCII são especialmente relevantes para o problema de compressão de textos:

- um código ASCII ocupa um byte de memória (ou oito bits), pois 255 é o maior número que pode ser armazenado em oito bits (11111111, em binário);
- o código ASCII possui mais códigos numéricos do que caracteres para representar (e, por isso, é chamado *degenerado*). Poderia, portanto, ser diminuído, sem muito sacrifício.

Na realidade, o ASC II é uma evolução de sistemas de códigos anteriormente existentes, que ocupavam menor espaço de memória, pois tinham sete ou seis bits (como o código Baudot, usado em máquinas de telex). A ampliação do espaço ocupado para oito bits significou uma adaptação ao mundo dos computadores digitais, onde predominam os processadores com número de bits organizados em potências de 2 — quatro, oito dezesseis, 32 bits etc. Com isso, expandiu-se no ASCII o alcance original dos códigos de texto.

Se os códigos dos caracteres fossem comprimidos em menos de oito bits, o espaço total para a armazenagem de texto na memória seria reduzido proporcionalmente. Podemos fazer isso adotando em nosso microcomputador um código diferente do ASCII e escrevendo um programa que faça a tradução do ASCII para o nosso código particular e vice-versa. Essa tradução é necessária porque os periféricos de entrada e saída, assim como as funções de programação que trabalham com textos, obedecem ao padrão ASCII. Existem diversas alternativas para a elaboração de um programa desse tipo, cada uma com suas vantagens e desvantagens.

DE OITO PARA SEIS

A maneira mais direta e intuitiva de se comprimir um texto consiste em utilizar apenas uma parte do código ASCII. De modo geral, qualquer texto pode ser escrito com o emprego de apenas 128 códigos, que cabem em sete bits. A economia de espaço obtida, nesse caso, é da ordem de 1 em 8, ou, em termos percentuais, de 12,5%.

É possível também escrever um texto empregando caracteres maiúsculos, numerais e sinais de pontuação, que correspondem aos códigos ASCII 32 a 90, ou seja, a 59 códigos. Se usarmos seis bits (o que permite armazenar um máximo de 64 códigos), conseguiremos uma redução de 2 em 8, ou de 25%, o que é bem razoável para muitas aplicações.

Converter o código ASCII ao nosso novo código de seis bits é fácil: basta diminuir 32 do código ASCII. O resultado será um código com valores de 0 a 58, com os caracteres representados exatamente na mesma ordem. Para a conversão no sentido inverso, basta somar 32 — e teremos o código ASCII, de novo.

- ARMAZENAGEM DO TEXTO
- ECONOMIA DE ESPAÇO
- TEORIA DA COMPRESSÃO
- CÓDIGO ASCII
- O USO DA ESTATÍSTICA

A redução pode ser ainda maior. Um código de cinco bits permite que se faça a representação de 32 caracteres: todas as letras maiúsculas, que são 26, e mais seis caracteres destinados à pontuação — por exemplo, o espaço em branco, o ponto, a vírgula, os dois pontos, o hífen e o sinal de interrogação. Se o texto contiver números, eles terão que ser escritos por extenso, o que, na verdade, não representa um problema muito grande.

Com um código de cinco bits, conseguimos uma redução de 5 em 8, ou seja, de 37,5%, o que significaria uma economia de quase quatro Kbytes em cada dez Kbytes de texto. Uma redução, convenhamos, considerável.

UM PROGRAMA DE CONVERSÃO

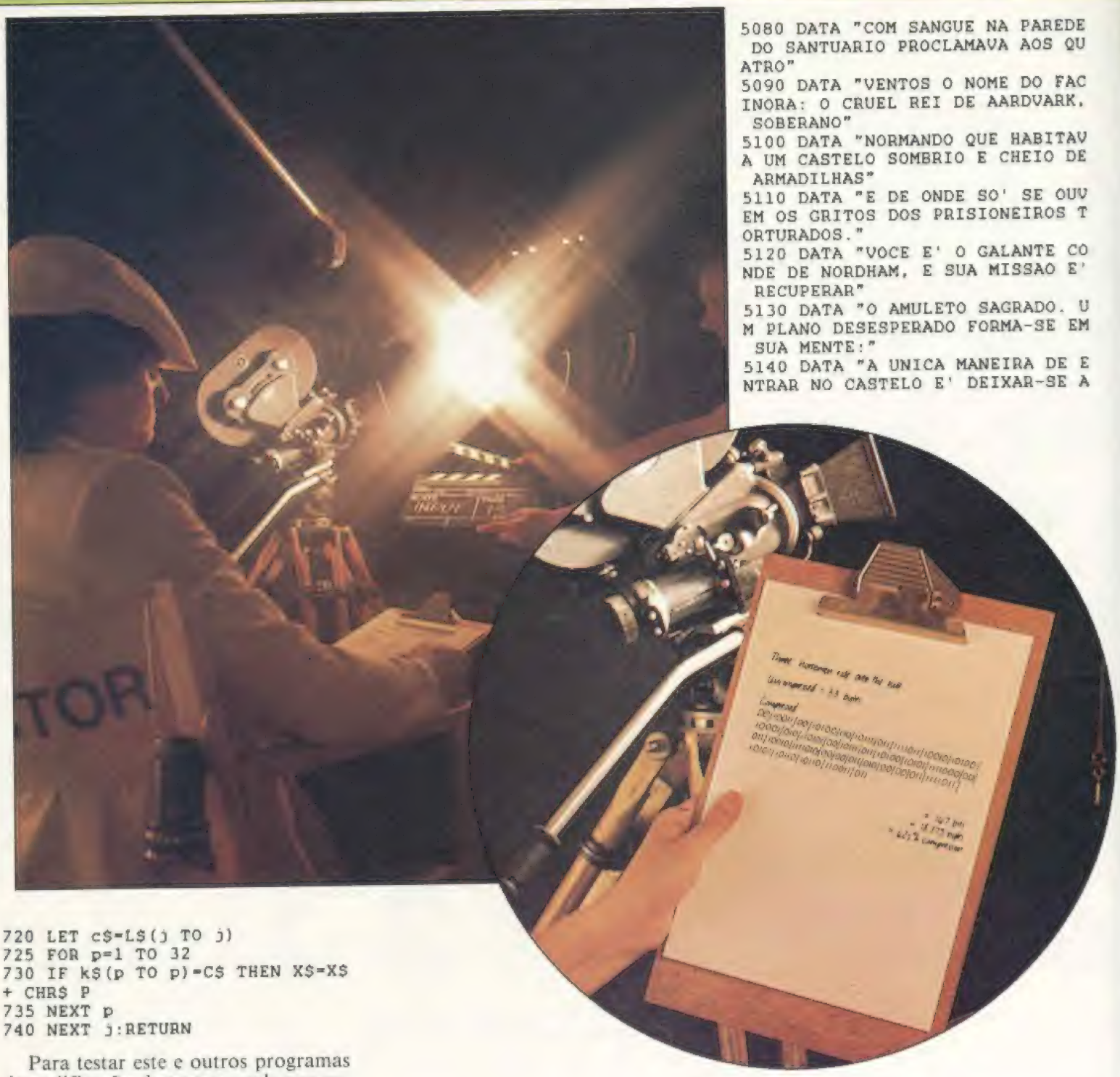
A conversão do ASCII e para o ASCII, nesse caso, não é tão fácil assim, exigindo um programa específico, como o que se segue:

```
10 DIM F$(50)
20 K$="ABCDEFGHIJKLMNOPQRSTUVW
XYZ ,.:-?"
100 CLS
180 PRINT "CODIFICANDO..."
190 I=0:NC=0
200 READ L$:IF L$="*" THEN 230
210 GOSUB 710:I=I+1:F$(I)=X$
215 NC=NC+LEN(X$)
220 GOTO 200
230 STOP
700 REM --- CODIFICACAO
710 X$="":FOR J=1 TO LEN(L$)
720 C$=MID$(L$,J,1)
730 P=INSTR(K$,C$):IF P>0 THEN
X$=X$+CHR$(P)
740 NEXT J:RETURN
```

Coloque o comando **CLEAR 1000** na linha 10, para micros das linhas TRS-80 e TRS-Color.

```
10 DIM F$(50)
20 K$="ABCDEFGHIJKLMNOPQRSTUVW
   XYZ ,.:-?"
100 HOME
180 PRINT "CODIFICANDO..."
190 LET I=0:NC=0
200 READ L$:IF L$="*" THEN 230
210 GOSUB 710:I=I+1:F$(I)=X$
215 LET NC=NC+LEN(X$)
220 GOTO 200
230 STOP
700 REM --- CODIFICACAO
710 LET X$="":FOR J=1 TO
    LEN(L$)
720 LET C$=MID$(L$,J,1)
725 FOR P=1 TO 32
730 IF MID$(K$,P,1)=C$ THEN X$
    = X$ + CHR$(P)
735 NEXT P
740 NEXT J:RETURN
```

```
10 DIM f$(50,64),k$(32)
20 LET k$="ABCDEFGHIJKLMNOPQRST
UVWXYZ ,.:-?"
180 PRINT "CODIFICANDO..."
190 RESTORE:LET i=0:LET nc=0
200 READ L$:IF L$="*" THEN GOTO
230
210 GOSUB 710:LET i=i+1:LET
f$(I)=x$
215 LET nc=nc+LEN x$
220 GOTO 200
230 STOP
700 REM --- CODIFICACAO
710 LET x$="":FOR j=1 TO LEN L$
```



```
720 LET c$=L$(j TO j)
725 FOR p=1 TO 32
730 IF k$(p TO p)=C$ THEN X$=X$
+ CHR$ P
735 NEXT p
740 NEXT j:RETURN
```

Para testar este e outros programas de codificação de textos, podemos colocar no fim do programa várias linhas **DATA**, contendo, por exemplo, as instruções de um jogo de aventura (a última linha deve ter um asterisco):

```
5000 DATA "POR MAIS DE DOIS MIL
 ANOS, O AMULETO SAGRADO DE NIT
PU FOI"
5010 DATA "ZELOSAMENTE GUARDADO
 PELOS ALDEAES DO CONDADO DE NO
RDHAM."
5020 DATA "A SUA POSSESSAO ERA
A CHAVE PARA A SEGURANCA E A FE
LICIDADE"
5030 DATA "DO REINO DE DUCHESS,
 NA ESCOCIA. O CONDE, QUE ERA O
 PROPRIETARIO"
5040 DATA "DE TODO AS TERRAS AO
 REDOR DA ALDEIA, JUROU DEFENDE
R O AMULETO"
5050 DATA "COM SUA PROPRIA VIDA
 "
5060 DATA "UM DIA, HORROR E DES
GRACA SE ABATEM SOBRE A ALDEIA.
 O"
5070 DATA "AMULETO SAGRADO TINH
A DESAPARECIDO ! UMA MENSAGEM E
SCRITA"
5080 DATA "COM SANGUE NA PAREDE
 DO SANTUARIO PROCLAMAVA AOS QU
ATRO"
5090 DATA "VENTOS O NOME DO FAC
INORA: O CRUEL REI DE AARDVARK,
 SOBERANO"
5100 DATA "NORMANDO QUE HABITAV
A UM CASTELO SOMBRIO E CHEIO DE
 ARMADILHAS"
5110 DATA "E DE ONDE SO' SE OUV
EM OS GRITOS DOS PRISIONEIROS T
ORTURADOS."
5120 DATA "VOCE E' O GALANTE CO
NDE DE NORDHAM, E SUA MISSAO E'
 RECUPERAR"
5130 DATA "O AMULETO SAGRADO. U
M PLANO DESESPERADO FORMA-SE EM
 SUA MENTE:"
5140 DATA "A UNICA MANEIRA DE E
NTRAR NO CASTELO E' DEIXAR-SE A
PRISIONAR"
5150 DATA "PELOS ASSECLAS FEROZ
ES DE AARDVARK. PARA COMPLETAR
A AVENTURA"
5160 DATA "VOCE PRECISA SE DESV
ENCILHAR DE SEUS CAPTORES, PERC
ORRER OS"
5170 DATA "LABIRINTOS MORTAIS D
O CASTELO, ACHAR O AMULETO, E D
EPOIS"
5180 DATA "ESCAPAR."
5190 DATA "SEUS RECURSOS SAO PO
UCOS: UMA ESPADA, UMA TOCHA, UM
 PUNHADO"
5200 DATA "DE ALIMENTOS SECOS,
```

```
UM CANTIL D'AGUA, E UM PUNHAL.
A SUA FRENTE,"
5210 DATA "UMA SEQUENCIA DE BAN
DIDOS E FERAS DE ARREPIAR OS CA
BELOS..."
5220 DATA "BOA SORTE !"
5230 DATA "*"
```

Se preferir, digite apenas as primeiras linhas. Só precisaremos deste texto completo quando formos testar programas mais complexos, adiante.

COMO FUNCIONA

O programa de codificação, que transforma o código ASCII em um código reduzido de cinco bits, está contido no laço das linhas 200 a 220. A linha 200 lê uma linha de texto, **L$**, em **DATA**, terminando ao encontrar um asterisco. A linha 710, por sua vez, chama a sub-rotina de codificação, que verifica se cada caractere **C$**, extraído de **L$**, está na lista de codificação armazenada em **K$** pela linha 20 do programa principal. Se estiver, sua posição seqüencial **P**, em **K$**, é usada como código, e um **CHR$(P)** concatena este caractere em uma cadeia de saída, **X$**. Se não estiver, então nada é concatenado.

O conteúdo de **X$** é armazenado na lista **F$**, onde cada elemento corresponde a uma linha do texto em **DATA**. A armazenagem é necessária para uso posterior. A variável **NC** da linha 225 serve simplesmente para contar o número de caracteres convertidos.

Como os textos armazenados em **F$** já não se encontram mais em ASCII, não podemos listá-los com um **PRINT** normal, pois muitos deles correspondem a caracteres ASCII de controle e provocariam a maior desordem na tela.

Se quiser ver como ficam os códigos numéricos convertidos, acrescente estas linhas ao programa e rode-o:

```
212 GOSUB 820
800 REM --- LISTAGEM
820 FOR N=1 TO LEN(X$)
830 PRINT ASC(MID$(X$,N,1));
840 NEXT N:PRINT
850 RETURN
```

```
212 GOSUB 820
800 REM --- LISTAGEM
820 FOR n=1 TO LEN x$
830 PRINT ASC X$(n TO n);
840 NEXT n:PRINT
850 RETURN
```

Para decodificar os textos — ou para reconvertê-los ao código ASCII —, devemos adicionar as linhas seguintes e rodar novamente o programa:

```
230 NL=I:T=0
235 PRINT "TOTAL: ";NC; "CARACT
ERES":PRINT"DECODIFICANDO..."
240 FOR I=1 TO NL
250 L$=F$(I):GOSUB 870
260 T=T+1:IF T<10 THEN 290
270 PRINT:INPUT"PRESSIONE <ENTE
R>";X$
280 T=0
290 NEXT I
300 STOP
860 REM --- DECODIFICACAO
870 FOR J=1 TO LEN(L$)
880 C=ASC(MID$(L$,J,1))
920 PRINT MID$(K$,C,1);
940 NEXT J
950 PRINT:RETURN
```

```
230 LET nl=I:LET t=0
235 PRINT "TOTAL: ";nc; "CARACT
ERES":PRINT"DECODIFICANDO..."
240 FOR i=1 TO nl
250 LET L$=f$(i) : GOSUB 870
260 LET t=t+1:IF t<10 THEN GOTO
290
270 PRINT:INPUT "PRESSIONE <ENT
ER>";x$
280 LET t=0
290 NEXT i
300 STOP
860 REM --- DECODIFICACAO
870 FOR j=1 TO LEN L$
880 LET c=ASC L$(j TO j)
920 PRINT K$(c TO c);
940 NEXT j
950 PRINT:RETURN
```

A rotina de decodificação (da linha 870 à 950) extrai cada caractere do texto codificado e troca seu código pelo caractere encontrado na mesma posição, na chave de codificação **K$**.

MANIPULAÇÃO DE BITS

Ainda não ganhamos espaço na memória: apesar de termos reduzido o conjunto de caracteres usados, eles continuam sendo armazenados em um byte (desperdiçando três bits em cada byte).

Para comprimir o texto de fato, precisamos utilizar os bits que estão sobrando, no seguinte esquema:

```
1º byte - 5 bits do 1º caractere
          3 bits do 2º caractere
2º byte - 2 bits do 2º caractere
          5 bits do 3º caractere
          1 bit  do 4º caractere
3º byte - 4 bits do 4º caractere
          4 bits do 5º caractere
```

e assim por diante.

Esta tarefa de manipulação de bits pode ser cumprida sem grande dificuldade em linguagem BASIC — por meio dos operadores lógicos **AND**, **OR** e **NOT** —, porém, é lenta demais. Na verdade, o ideal seria realizá-la através de uma sub-rotina **USR** em linguagem de máquina, mas não vamos fazê-lo aqui, para não tornar muito complicada a programação do jogo de aventura.

Optamos por um outro método de compressão, também baseado em manipulação de bits, mas que pode ser facilmente executado em BASIC. Esse método é bem mais eficaz que a restrição do conjunto de caracteres, pois garante uma redução do texto de cerca de 45%.

A programação, aqui, é mais simples, pois os bits não são manipulados individualmente, mas em grupos de quatro (meio byte ou *nibble*, em inglês). Com uma única expressão aritmético-lógica simples podemos ter acesso ou escrever nos nibbles individuais.

UM POUCO DE ESTATÍSTICA

Como sabemos, cada idioma utiliza determinadas letras com maior freqüência do que outras. A técnica que escolhemos fundamenta-se exatamente na freqüência do emprego das letras. Na língua portuguesa, por exemplo, catorze letras — A, S, E, D, R, O, I, U, N, C, M, T, P e L — representam cerca de 90% do total de um texto, juntamente com o espaço em branco.

Usaremos um código baseado apenas em quatro bits (15 é o maior número decimal que se pode representar com quatro bits, ou um nibble). Com isso, obteremos uma compressão de 50% do texto, o que é, teoricamente, o máximo que se pode conseguir. E será fácil fazer o programa, porque caberão sempre dois caracteres por byte.

Mas o que fazer com as outras letras? Mesmo sendo usadas com uma freqüência muito baixa, elas são essenciais para compressão do texto. Precisam, por isso, estar presentes.

A solução também é espantosamente simples: podemos usar essas letras com seu código ASCII normal, de oito bits. No processo de codificação, toda vez que o programa encontrar uma letra não incluída no grupo das mais freqüentes, ele atribuirá um código 0 a ela. Dessa maneira, indica-se ao programa decodificador que o que vem a seguir é um código ASCII de oito bits.

Uma letra incomum usa, assim, três nibbles (oito do código, mais quatro da baliza 0), ou um byte e meio: este é o preço que se paga por codificar a maio-

ria das letras com quatro bits. A eficiência da compressão será menor, mas não muito. Façamos as contas, tomando um conjunto de cem caracteres:

90 têm códigos de 4 bits = 45 bytes
10 têm códigos de 12 bits = 15 bytes

TOTAL = 60 bytes

Conseguimos, teoricamente, uma compressão de 40%. Na realidade, ela será um pouco menor, dependendo do texto. Ainda assim, o resultado é bem melhor do que o obtido por uma compressão com códigos de seis bits.

Para verificar com que freqüência são empregadas as letras presentes em nosso texto-teste, digite o programa que se segue, acrescente as linhas **DATA** ao final e rode-o:

```
15 DIM F(60),C(60)
30 NT=0
40 FOR I=1 TO 60
50 F(I)=0:C(I)=I
60 NEXT I
65 PRINT "ANALISANDO..."
70 READ L$:IF L$="*" THEN GOTO
85
80 GOSUB 410:GOTO 70
85 PRINT "ORDENANDO..."
90 GOSUB 470:GOSUB 560
100 STOP
400 REM --- CONTAGEM
410 FOR I=1 TO LEN(L$)
420 NT=NT+1
430 J=ASC(MID$(L$,I,1))-31
440 F(J)=F(J)+1:NEXT I
450 RETURN
460 REM --- ORDENACAO
470 N=60
480 FL=0
490 N=N-1:FOR I=1 TO N
500 IF F(C(I))=>F(C(I+1)) THEN
520
510 X=C(I):C(I)=C(I+1):C(I+1)=X
:FL=1
520 NEXT I
530 IF FL=0 OR N=2 THEN RETURN
540 GOTO 480
550 REM --- IMPRESSAO
560 PRINT
570 PRINT "FREQUENCIA SIMPLES
DOS CARACTERES NO TEXTO":PRINT
580 FOR J=1 TO 60 STEP 3
590 FOR I=J TO J+2
600 IF F(C(I))=0 THEN GOTO 640
610 PRINT CHR$(C(I)+31);" " ;
F(C(I)),
620 NEXT I:PRINT
630 NEXT J
640 PRINT:PRINT"TOTAL: ";NT;"
CARACTERES."
650 RETURN
```

Coloque um comando **CLEAR 3000** na linha 15, para os micros pertencentes às linhas TRS-80 e TRS-Color.

```
15 DIM f(60),c(60)
30 LET nt=0
40 FOR i=1 TO 60
50 LET f(i)=0:LET c(i)=i
60 NEXT i
65 PRINT "ANALISANDO..."
70 READ L$:IF L$="*" THEN GOTO
85
80 GOSUB 410:GOTO 70
85 PRINT "ORDENANDO..."
90 GOSUB 470:GOSUB 560
100 STOP
400 REM --- CONTAGEM
410 FOR i=1 TO LEN L$
420 LET nt=nt+1
430 LET j=ASC(L$(i to i))-31
440 LET f(j)=f(j)+1:NEXT i
450 RETURN
460 REM --- ORDENACAO
470 LET n=60
480 LET f1=0
490 LET n=n-1:FOR i=1 TO n
500 IF f(c(i))=>f(c(i+1)) THEN
GOTO 520
510 LET x=c(i):LET c(i)=c(i+1)
: c(i+1)=x LET f1=1
520 NEXT i
530 IF f1=0 OR n=2 THEN RETURN
540 GOTO 480
550 REM --- IMPRESSAO
560 PRINT
570 PRINT "FREQUENCIA SIMPLES
DOS CARACTERES NO TEXTO":PRINT
580 FOR j=1 TO 60 STEP 3
590 FOR i=j TO j+2
600 IF f(c(i))=0 THEN GOTO 640
610 PRINT CHR$(c(i)+31);" " ;
f(c(i)),
620 NEXT i:PRINT
630 NEXT j
640 PRINT:PRINT"TOTAL: ";nt;"
CARACTERES."
650 RETURN
```

As linhas 15 a 60 definem e inicializam dois conjuntos: **F**, que conterá a freqüência de cada caractere presente no texto, e **C**, um conjunto-índice que será utilizado pela rotina de ordenação (no começo **C** contém os números inteiros de 1 a 60, em ordem crescente). A variável **NT** servirá para contar o número total de caracteres no texto.

O laço que abrange as linhas 70 e 80 lê as linhas de texto original em **DATA**, parando se encontrar um asterisco. Caso contrário, chama a rotina 410, que tem por objetivo executar a contagem. Essa rotina simplesmente acha o código ASCII de cada um dos caracteres presentes na linha de texto, diminui o valor 31 dos mesmos, e incrementa o elemento de **F** correspondente.

Em seguida, as sub-rotinas 470 e 560 (chamadas na linha 90 do programa) encarregam-se, respectivamente, de colocar o conjunto **C** em ordem decrescente de freqüência e mostrar o resultado na tela. A rotina de ordenação usa o método tipo bolha, que examinamos no artigo da página 468. A rotina de exibição mostra apenas os caracteres com freqüência maior do que 0.

Se tudo tiver corrido bem, você obterá uma tabela com estes dados:

```
   192     A  147     E  129
O  111     S   87     R   84
D   66     I   47     U   45
N   43     C   40     M   39
T   36     P   30     L   29
,   15     H   14     .   12
V   12     G   11     B    9
F    8     '    5     Q    4
:    3     !    2     -    2
K    2     Z    2     J    1
X    1

TOTAL:   1228   CARACTERES.
```

Note que, como afirmamos anteriormente, um conjunto de apenas catorze letras mais o espaço em branco respondem por 91,6% do texto (1125 ocorrências em 1228 letras). O espaço em branco é o caractere mais freqüente em qualquer texto, por razões óbvias.

Um segundo conjunto, formado por mais catorze caracteres, responde por 8% do texto. Caberia a um terceiro conjunto abrigar os demais caracteres (a maioria dos quais não apareceu nenhuma vez neste texto, mas poderia constar esporadicamente de outros).

Vimos que, se usarmos apenas um conjunto de codificação com os quinze caracteres mais freqüentes, teremos que colocar os restantes no texto sem codificar, e cada um deles ocupará três nibbles. Entretanto, podemos estender a idéia de um conjunto de quatro bits de código para dois ou mais conjuntos adicionais. Assim, a codificação de cada caractere constante do segundo conjunto ocupará dois nibbles (um com o 0 e outro com o código propriamente dito); a codificação dos caracteres do terceiro conjunto ocupará três nibbles, e assim por diante. Vamos fazer as contas de novo, para o mesmo conjunto de cem caracteres:

90 têm códigos de 4 bits = 45 bytes
9 têm códigos de 8 bits = 9 bytes
1 tem código de 12 bits = 2 bytes

TOTAL = 56 bytes

Como você pode notar, conseguimos aumentar nossa eficiência teórica de compressão para 44% (o que representa uma melhoria de 10% em relação ao esquema discutido anteriormente).

Podemos agora desenvolver um programa de codificação (acrescente as linhas **DATA** de teste ao final):

```
10 DIM F$(50),K$(3)
20 K$(1)=" AEOSRDIUNCMTPL"
25 K$(2)=",H.VGBF'Q:!-KZJ"
27 K$(3)="XWY;=/*?()$%+"
180 PRINT "CODIFICANDO..."
190 RESTORE:I=0:NC=0:K=1
200 READ L$:IF L$="*" THEN 230
210 GOSUB 710:I=I+1:F$(I)=X$
212 GOSUB 820
215 NC=NC+LEN(X$)
220 GOTO 200
230 NL=I:T=0
235 PRINT "TOTAL: " ; NC ;
"CARACTERES"
237 STOP
700 REM --- CODIFICACAO
710 N=0:X$="":Y$="":L$=L$+" "
720 FOR J=1 TO LEN(L$)
730 C$=MID$(L$,J,1)
740 P=INSTR(K$(K),C$)
760 N=N+1:IF N=1 THEN B=P
770 IF N=2 THEN N=0:B=B OR
(16*P) : X$=X$+CHR$(B)
775 IF P=0 THEN K=K+1:GOTO 740
776 K=1
780 NEXT J
790 RETURN
```

Coloque um comando **CLEAR 3000** na linha 10, para os micros pertencentes às linhas TRS-80 e TRS-Color.

Para que o programa funcione nos microcomputadores Apple e TK-2000, substitua a rotina de codificação das linhas 710 a 790 por:

```
700 REM --- CODIFICACAO
710 N=0:X$="":L$=L$+" "
720 FOR J=1 TO LEN(L$)
730 C$=MID$(L$,J,1)
735 FOR P=1 TO LEN(K$(K))
740 IF MID$(K$(K),P,1) = C$
    THEN 760
750 NEXT P
760 N=N+1:IF N=1 THEN B=P
770 IF N=2 THEN N=0:B=B OR
    (16*P) : X$=X$+CHR$(B)
775 IF P=0 THEN K=K+1:GOTO 740
776 K=1
780 NEXT J
790 RETURN
```

```
10 DIM f$(50,64),k$(3)
20 k$(1)=" AEOSRDIUNCMTPL"
25 k$(2)=",H.VGBF'Q:!-KZJ"
27 k$(3)="XWY;=/*?()$%+"
180 PRINT "CODIFICANDO..."
190 RESTORE:i=0:nc=0:k=1
200 READ L$:IF L$="*" THEN GOTO
230
210 GOSUB 710:LET i=i+1:LET
f$(i)=X$
212 GOSUB 820
215 LET nc=nc+LEN x$
220 GOTO 200
230 LET n1=I:LET t=0
235 PRINT "TOTAL: ";nc ;
"CARACTERES"
237 STOP
700 REM --- CODIFICACAO
710 LET n=0:LET x$="":LET
L$=L$+" "
720 FOR j=1 TO LEN L$
730 LET c$=L$(j TO j)
735 FOR p=1 TO LEN k$(k)
740 IF k$(K,p TO p) = c$ THEN
GOTO 760
750 NEXT p
760 LET n=n+1:IF n=1 THEN LET
b=p
770 IF n=2 THEN LET n=0:LET b=b
OR (16*p) : LET x$=x$+CHR$ b
775 IF p=0 THEN LET k=k+1:GOTO
740
776 LET k=1
780 NEXT j
790 RETURN
```

As linhas 20 a 27 definem os três conjuntos de caracteres a serem usados. O primeiro conjunto reúne os quinze caracteres mais freqüentes em um texto. Para simplificar, utilizaremos a mesma ordem encontrada no texto de teste. Mas, se quiséssemos obter um conjunto padrão, que servisse igualmente bem para qualquer texto na língua portuguesa, deveríamos analisar uma amostra muitas vezes mais extensa antes de definir a ordem dos caracteres. O segundo conjunto contém os próximos quinze caracteres de maior freqüência no texto e assim por diante.

A seção do programa que abrange as linhas 200 a 220 lê o texto contido em **DATA** e chama a rotina de codificação, que tem início na linha 710. O texto codificado, devolvido a **X$**, é armazenado no conjunto **F$**, para posterior uso na decodificação.

ROTINA DE CODIFICAÇÃO

Por ser um tanto complexa, a rotina de codificação exige uma explicação mais detalhada. Ela procura, no primeiro conjunto de códigos (que está em **K$(1)**), cada um dos caracteres individuais da linha de texto, extraídos e guardados em **C$**. Não encontrando ali determinado caractere, o apontador **K** é incrementado e a rotina passa a procurá-lo no segundo conjunto, e assim por diante. Todas as vezes que passa para outro conjunto de códigos, a rotina coloca um código 0 na seqüência de saída. Assim, um caractere encontrado no segundo conjunto vai ter um 0 seguido de seu código, e um caractere encontrado no terceiro conjunto terá dois zeros antes de seu código.

A cada dois códigos numéricos gerados (contados pela variável **N**), as linhas 760 e 770 da rotina comprimem os dois nibbles em um byte, **B**, através da expressão matemática da linha 770. Vejamos um exemplo:

	decimal	**binário**
1º nibble	14	00001110
2º nibble	7	00000111

Expressão: 14 **OR** (16*7)
14 **OR** 112

00001110 **OR** 01110000

que é igual a 01111110 ou 126.

Note que a multiplicação de um nibble por 16 tem o efeito de deslocá-lo da parte baixa para a parte alta do byte. Uma operação **OR** soma o primeiro nibble (não deslocado) com o segundo (deslocado); o resultado é um byte com ambos os nibbles em cada metade.

Se você quiser acompanhar a geração de códigos comprimidos, acrescente as linhas que se seguem e rode o programa novamente.

```
212 GOSUB 820
800 REM --- LISTAGEM
820 FOR N=1 TO LEN(X$)
830 PRINT ASC(MID$(X$,N,1));
840 NEXT N:PRINT
850 RETURN
```

```
212 GOSUB 820
800 REM --- LISTAGEM
820 FOR n=1 TO LEN x$
830 PRINT ASC (X$(n TO n));
840 NEXT n:PRINT
850 RETURN
```

Para obter o texto original de volta, basta que você acrescente a rotina de decodificação:

```
237 PRINT "DECODIFICANDO..."
240 FOR I=1 TO NL
250 L$=F$(I):GOSUB 870
260 T=T+1 :IF T<15 THEN GOTO 290
270 PRINT:INPUT "PRESSIONE <EN
TER>";X$
280 T=0
290 NEXT I
300 STOP
860 REM --- DECODIFICACAO
870 N=0:K=1:FOR J=1 TO LEN(L$)
880 C=ASC(MID$(L$,J,1))
890 C(1)=C AND 15:C(2)=(C AND
240)/16
```

```
900 FOR L=1 TO 2
910 IF C(L)=0 THEN K=K+1:GOTO
930
920 PRINT MID$(K$(K),C(L),1);
925 K=1
930 NEXT L
940 NEXT J
950 PRINT:RETURN
```

```
237 PRINT "DECODIFICANDO..."
240 FOR i=1 TO n1
250 LET L$=f$(i):GOSUB 870
260 LET t=t+1:IF t<15 THEN GOTO
290
270 PRINT:INPUT "PRESSIONE <ENT
ER> ";X$
280 LET t=0
290 NEXT i
300 STOP
860 REM --- DECODIFICACAO
870 LET n=0:LET k=1:FOR j=1 TO
LEN L$
880 LET c=ASC(L$(j TO j))
890 LET c(1)=c AND 15:c(2)=(c
AND 240)/16
900 FOR L=1 TO 2
910 IF c(L)=0 THEN LET
k=k+1:GOTO 930
920 PRINT K$(K,c(L) to c(L));
925 LET k=1
930 NEXT L
940 NEXT J
950 PRINT:RETURN
```

A sub-rotina responsável pela decodificação funciona de maneira exatamente inversa à rotina de codificação, que acabamos de examinar.

A variável **C** contém o código comprimido, extraído de cada um dos caracteres da linha codificada. A partir desta obtêm-se os dois nibbles, que são armazenados respectivamente em **C(1)** e **C(2)**, na expressão da linha 890, e examinados pelo laço das linhas 900 a 930. Se um dos nibbles é igual a 0, o apontador **K**, que inicialmente se encontrava em 1, é incrementado, passando a indicar o próximo conjunto de códigos. Finalmente, a linha 920 imprime na tela o caractere decodificado.

Tomemos o exemplo anterior para ver como funciona a decodificação:

byte = 01111110 = 126

1º nibble 126 **AND** 15

ou: 01111110 **AND** 00001111
ou: 00001110
ou: 14 em decimal

2º nibble (126 **AND** 240)/16

ou: 01111110 **AND** 11110000
ou: 01110000
ou: 112/16 = 7 (00000111)

Observe que a divisão por 16 tem o efeito contrário da multiplicação por 16 — ou seja, desloca o nibble alto para o nibble baixo.

GRAU DE EFICIÊNCIA

Obteremos os seguintes resultados com nosso texto de teste: depois de codificado, ele foi reduzido de 1228 para 674 caracteres, ou seja, houve uma redução de 45%. Pouquíssimos métodos de compressão conseguem tão alto grau de eficiência. Na realidade, se contarmos o número total de bytes gastos pelo programa e pela tabela de códigos **K$**, a redução global de espaço vai ser bem menor. Devemos ter em mente, porém, que a utilização de um algoritmo de compressão como este só faz sentido quando o texto a ser comprimido é extenso. Assim, o acréscimo de memória necessário para o programa e as tabelas é insignificante se comparado aos ganhos obtidos pela compressão.

LIÇÕES DA CRIPTOGRAFIA

O compressor de textos desenvolvido aqui toma emprestado alguns conceitos da criptografia, que é a ciência das cifras e dos códigos (veja artigos das páginas 888 e 1091). Ao tentar decifrar um código secreto, o criptógrafo analisa, antes de mais nada, as freqüências dos caracteres presentes no texto cifrado. Estas podem dizer muito sobre a freqüência das letras usadas no idioma original.

O uso das letras mais freqüentes do alfabeto no primeiro conjunto de codificação assegura a obtenção do máximo de compressão possível. Se esse conjunto incluir caracteres que aparecem com menor freqüência, omitindo outros mais utilizados, a compressão perderá muito em eficiência, pois muitos códigos 0 serão gerados.

COMBINANDO ETAPAS

Normalmente, um programa compressor baseado nessa técnica deveria ser independente do texto que se deseja comprimir — ou seja, teria nos conjuntos **K$** uma seqüência padronizada de caracteres mais freqüentes, identificados pela análise de uma grande amostra da língua portuguesa.

Entretanto, um texto de aventura apresenta algumas peculiaridades, incluindo termos estranhos, como Aardvark, Niptu, Nordham etc., que não pertencem, evidentemente, à nossa língua. Por isso, verificou-se uma freqüência atípica da letra K.

Como precisamos codificar o texto de uma aventura apenas uma vez, nada impede que combinemos os programas de análise de freqüências e de codificação em um só, para otimizar o processo. Os programas anteriores foram escritos com números diferentes de linhas, de modo a facilitar essa integração. Para proceder à fusão dos dois programas, faça as modificações:

```
565 K=1
615 K$(K)=K$(K)+
CHR$(C(I)+31)
616 IF LEN(K$(K))=15
THEN K=K+1
```

e suprima as linhas 20 a 27.

```
565 LET k=1
615 LET k$(k)=k$(k)+CHR$(c
(i)+31)
616 IF LEN(k$(k))=15 THEN LET
k=k+1
```

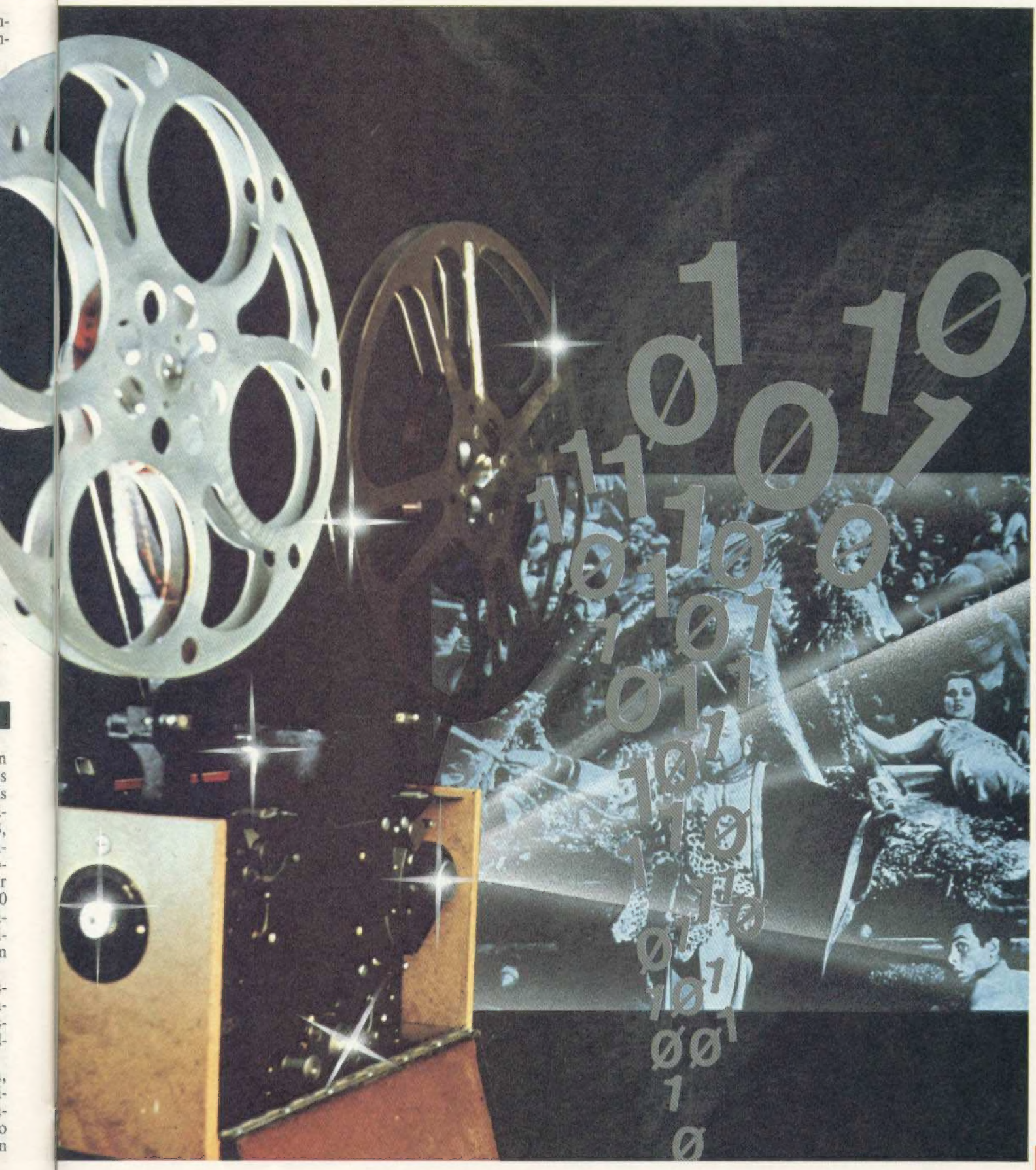
e suprima as linhas 20 a 27.

ELEMENTOS DE K$

Observe que algumas linhas foram acrescentadas à rotina de impressão dos resultados da análise de freqüência. Elas se encarregam de construir automaticamente os elementos do conjunto **K$**, com a vantagem de criar tantos elementos quantos se façam necessários (se forem mais de três, é conveniente alterar a dimensão deste conjunto na linha 10 do programa principal). Por isso, as linhas 20 a 27, que contêm a definição fixa do conteúdo do conjunto **K$**, devem ser retiradas.

Tente codificar outros textos com esse programa, e veja como a taxa de compressão se mantém quase a mesma. Esta é uma característica desejável de qualquer sistema de compressão.

No próximo artigo sobre este tema, investigaremos outras alternativas utilizadas na compressão de textos, particularmente o *Método Chinês*, por meio do qual é possível obter 60% de redução em uma aventura.



OS SEGREDOS DO SPECTRUM (2)

- VARIÁVEIS DO SISTEMA
- TIPOS DE VARIÁVEL
- COMO MODIFICAR
- APONTADORES E CONTADORES
- ENDEREÇOS ÚTEIS

Para o programador ambicioso, existe um mapa de valor inestimável: os endereços das variáveis do sistema. Com PEEK e POKE, você terá acesso a ele e obterá maravilhas com o micro.

O BASIC do Spectrum possui um grande número de comandos relativamente poderosos para trabalho com a tela, teclado, memória etc. Apesar disso, muitas propriedades da máquina e de seu sistema operacional ficam fora do alcance do programador que se restringe apenas a esses comandos.

Usando a função **PEEK** para leitura direta de uma locação absoluta de memória, e o comando **POKE** para modificar seu conteúdo, você poderá redefinir muitos parâmetros de funcionamento do sistema operacional e do interpretador BASIC no Spectrum.

Mas, se o sistema operacional e o interpretador estão gravados permanentemente na memória ROM da máquina, como é possível alterar o seu funcionamento? Quando o computador é ligado, o interpretador BASIC especifica uma área de trabalho na RAM, que vai da locação 23522 à 23732. Uma área adicional — de 23734 à 23791 — é criada, se o computador for conectado a uma interface universal de entrada/saída.

Cada locação nessas áreas armazena um valor numérico, usado pelo sistema operacional e pelo interpretador durante a execução de programas. Essas locações são chamadas *variáveis do sistema* e recebem um nome simbólico, para fins de referência. Podem ser:

- registros intermediários de operações, tais como: código da última tecla pressionada (**LASTK**), último código de erro encontrado (**ERRNR**) etc;
- parâmetros de operação de comandos: duração do bipe (**RASP**), atraso entre repetições de uma tecla (**REPPER**), cor da moldura (**BORDCR**) etc;
- indicadores: próxima linha a ser executada (**NEWPPC**), coordenada X do cursor de alta resolução (**COORDX**), número de linha da posição corrente de **PRINT** (**SPOSLN**) etc;
- apontadores, que contêm outros endereços de memória: início de um programa BASIC (**PROG**), endereço do cursor na RAM da tela (**KCUR**) etc;
- parâmetros de definição do hardware: topo da memória RAM (**PRAMT**) etc;
- contadores e acumuladores: contador de quadros de TV (**FRAMES**), espaço ocupado por cadeias (**STRLEN**) etc.

Algumas variáveis do sistema ocupam um byte, outras, dois. Quando ocupam dois bytes, o número de dezesseis bits nelas armazenado segue o esquema de dupla byte mais significativo/byte menos significativo.

VARIÁVEIS DO SISTEMA

Nome	Endereço	N.º de bytes	Utilização
LASTK	23560	1	Última tecla pressionada
REPDEL	23561	1	Tempo de pressão a uma tecla para repetir
REPPER	23562	1	Atraso entre repetições de uma tecla
RASP	23608	1	Duração do som de alarma
PIP	23609	1	Duração do "clique" de teclado
ERRNR	23610	1	Último código de erro, menos 1
MODE	23617	1	Modo do cursor (K, L, C, E ou G)
NEWPPC	23618	2	Próxima linha a ser executada
NSPPC	23620	2	Próximo comando da linha a ser executada
EPPC	23625	2	Número da linha com o cursor do programa
VARS	23627	2	Endereço do início da área de variáveis
PROG	23635	2	Endereço do início do programa BASIC
NXTLIN	23637	2	Endereço da próxima linha a executar
ELINE	23641	2	Endereço do *buffer* de entrada em BASIC
KCUR	23643	2	Endereço do cursor
FLAGS2	23658	1	Indicador de minúsculas/maiúsculas
DFSZ	23659	1	Número de linhas na zona inferior da tela
SEED	23670	2	Semente do gerador **RAND**
FRAMES	23672	3	Contador de quadros de TV
COORDX	23677	1	Coordenada X do último ponto gráfico
COORDY	23678	1	Coordenada Y do último ponto gráfico
PPOSN	23679	1	Coluna de impressão na impressora
FREE	23681	1	Byte livre para o usuário, não afetado por **NEW**
SPOSN	23688	1	Número de coluna da última posição **PRINT**
SPOSLN	23689	1	Número da linha da última posição **PRINT**
SCRCT	23692	1	Contador de linhas para o *scroll*
RAMTOP	23730	2	Topo da memória livre para BASIC
PRAMT	23732	2	Topo da memória RAM (física)

APLICAÇÕES

O acesso a esses endereços permitenos, entre outras coisas, copiar determinado valor em uma variável do sistema, de modo a informar o programa sobre algum parâmetro de interesse para o processamento subseqüente. Suponhamos que você queira dimensionar uma tabela em função do máximo de memória disponível no computador. Para isso, use o número de dezesseis bits armazenado no par 23730-23731, que é a variável de sistema **RAMTOP**. Ela indicará o topo máximo de memória RAM disponível para um programa em BASIC:

```
LET MAX = PEEK(23730)+PEEK(
23731)*256
```

Podemos também modificar um valor por meio de um **POKE**, fazendo com que dado comando seja executado de maneira diferente. Por exemplo: a locação 23692 corresponde à variável do sistema **SCRCT**, que é o contador de linhas usado pelo sistema para perguntar *scroll?* quando o cursor de texto chega ao fim da tela. Com um **POKE 23692,255** a cada número de linha, o computador nunca fará essa pergunta.

No quadro acima, listamos algumas das variáveis de sistema mais importantes para o Spectrum e compatíveis. Use a sua imaginação!

CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR LINHA

LINHA	FABRICANTE	MODELO
Apple II +	Appletronica	Thor 2010
Apple II +	CCE	MC-4000 Exato
Apple II +	CPA	Absolutus
Apple II +	CPA	Polaris
Apple II +	Digitus	DGT-AP
Apple II +	Dismac	D-8100
Apple II +	ENIAC	ENIAC II
Apple II +	Franklin	Franklin
Apple II +	Houston	Houston AP
Apple II +	Magnex	DM II
Apple II +	Maxitronica	MX-2001
Apple II +	Maxitronica	MX-48
Apple II +	Maxitronica	MX-64
Apple II +	Maxitronica	Maxitronic I
Apple II +	Microcraft	Craf II Plus
Apple II +	Milmar	Apple II Plus
Apple II +	Milmar	Apple Master
Apple II +	Milmar	Apple Senior
Apple II +	Omega	MC-400
Apple II +	Polymax	Maxxi
Apple II +	Polymax	Poly Plus
Apple II +	Spectrum	Microengenho I
Apple II +	Spectrum	Spectrum ed
Apple II +	Suporte	Venus II
Apple II +	Sycomig	SIC I
Apple II +	Unitron	AP II
Apple II +	Victor do Brasil	Elppa II Plus
Apple II +	Victor do Brasil	Elppa Jr.
Apple IIe	Microcraft	Craft IIe
Apple IIe	Microdigital	TK-3000 IIe
Apple IIe	Spectrum	Microengenho II
MSX	Gradiente	Expert GPC-1
MSX	Sharp	Hotbit HB-8000
Sinclair Spectrum	Microdigital	TK-90X
Sinclair Spectrum	Timex	Timex 2000
Sinclair ZX-81	Apply	Apply 300
Sinclair ZX-81	Engebras	AS-1000
Sinclair ZX-81	Filcres	NEZ-8000
Sinclair ZX-81	Microdigital	TK-82C
Sinclair ZX-81	Microdigital	TK-83
Sinclair ZX-81	Microdigital	TK-85
Sinclair ZX-81	Prologica	CP-200
Sinclair ZX-81	Ritas	Ringo R-470
Sinclair ZX-81	Timex	Timex 1000
Sinclair ZX-81	Timex	Timex 1500
TRS-80 Mod. I	Dismac	D-8000
TRS-80 Mod. I	Dismac	D-8001/2
TRS-80 Mod. I	LNW	LNW-80
TRS-80 Mod. I	Video Genie	Video Genie I
TRS-80 Mod.III	Digitus	DGT-100
TRS-80 Mod.III	Digitus	DGT-1000
TRS-80 Mod.III	Kemitron	Naja 800
TRS-80 Mod.III	Prologica	CP-300
TRS-80 Mod.III	Prologica	CP-500
TRS-80 Mod.III	Sysdata	Sysdata III
TRS-80 Mod.III	Sysdata	Sysdata Jr.
TRS-80 Mod.IV	Multix	MX-Compacto
TRS-80 Mod.IV	Sysdata	Sysdata IV
TRS-Color	Codimex	CS-6508
TRS-Color	Dynacom	MX-1600
TRS-Color	LZ	Color 64
TRS-Color	Microdigital	TKS-800
TRS-Color	Prologica	CP-400

CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR NOME

FABRICANTE	MODELO	PAÍS	LINHA
Appletronica	Thor 2010	Brasil	Apple II +
Apply	Apply 300	Brasil	Sinclair ZX-81
CCE	MC-4000 Exato	Brasil	Apple II +
CPA	Absolutus	Brasil	Apple II +
CPA	Polaris	Brasil	Apple II +
Codimex	CS-6508	Brasil	TRS-Color
Digitus	DGT-100	Brasil	TRS-80 Mod.III
Digitus	DGT-1000	Brasil	TRS-80 Mod.III
Digitus	DGT-AP	Brasil	Apple II +
Dismac	D-8000	Brasil	TRS-80 Mod. I
Dismac	D-8001/2	Brasil	TRS-80 Mod. I
Dismac	D-8100	Brasil	Apple II +
Dynacom	MX-1600	Brasil	TRS-Color
ENIAC	ENIAC II	Brasil	Apple II +
Engebras	AS-1000	Brasil	Sinclair ZX-81
Filcres	NEZ-8000	Brasil	Sinclair ZX-81
Franklin	Franklin	USA	Apple II +
Gradiente	Expert GPC1	Brasil	MSX
Houston	Houston AP	Brasil	Apple II +
Kemitron	Naja 800	Brasil	TRS-80 Mod.III
LNW	LNW-80	USA	TRS-80 Mod. I
LZ	Color 64	Brasil	TRS-Color
Magnex	DM II	Brasil	Apple II +
Maxitronica	MX-2001	Brasil	Apple II +
Maxitronica	MX-48	Brasil	Apple II +
Maxitronica	MX-64	Brasil	Apple II +
Maxitronica	Maxitronic I	Brasil	Apple II +
Microcraft	Craft II Plus	Brasil	Apple II +
Microcraft	Caft IIe	Brasil	Apple IIe
Microdigital	TK-3000 IIe	Brasil	Apple IIe
Microdigital	TK-82C	Brasil	Sinclair ZX-81
Microdigital	TK-83	Brasil	Sinclair ZX-81
Microdigital	TK-85	Brasil	Sinclair ZX-81
Microdigital	TK-90X	Brasil	Sinclair Spectrum
Microdigital	TKS-800	Brasil	TRS-Color
Milmar	Apple II Plus	Brasil	Apple II +
Milmar	Apple Master	Brasil	Apple II +
Milmar	Apple Senior	Brasil	Apple II +
Multix	MX-Compacto	Brasil	TRS-80 Mod.IV
Omega	MC-400	Brasil	Apple II +
Polymax	Maxxi	Brasil	Apple II +
Polymax	Poly Plus	Brasil	Apple II +
Prologica	CP-200	Brasil	Sinclair ZX-81
Prologica	CP-300	Brasil	TRS-80 Mod.III
Prologica	CP-400	Brasil	TRS-Color
Prologica	CP-500	Brasil	TRS-80 Mod.III
Ritas	Ringo R-470	Brasil	Sinclair ZX-81
Sharp	Hotbit HB-8000	Brasil	MSX
Spectrum	Microengenho I	Brasil	Apple II +
Spectrum	Microengenho II	Brasil	Apple IIe
Spectrum	Spectrum ed	Brasil	Apple II +
Suporte	Venus II	Brasil	Apple II +
Sycomig	SIC I	Brasil	Apple II +
Sysdata	Sysdata III	Brasil	TRS-80 Mod.III
Sysdata	Sysdata IV	Brasil	TRS-80 Mod.IV
Sysdata	Sysdata Jr.	Brasil	TRS-80 Mod.III
Timex	Timex 1000	USA	Sinclair ZX-81
Timex	Timex 1500	USA	Sinclair ZX-81
Timex	Timex 2000	USA	Sinclair Spectrum
Unitron	AP II	Brasil	Apple II +
Victor do Brasil	Elppa II Plus	Brasil	Apple II +
Victor do Brasil	Elppa Jr.	Brasil	Apple II +
Video Genie	Video Genie I	USA	TRS-80 Mod. I

UM LOGOTIPO PARA CADA MODELO DE COMPUTADOR

INPUT foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado.
Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:

Sinclair ZX-81

TRS-80

TK-2000

MSX

Spectrum

TRS-Color

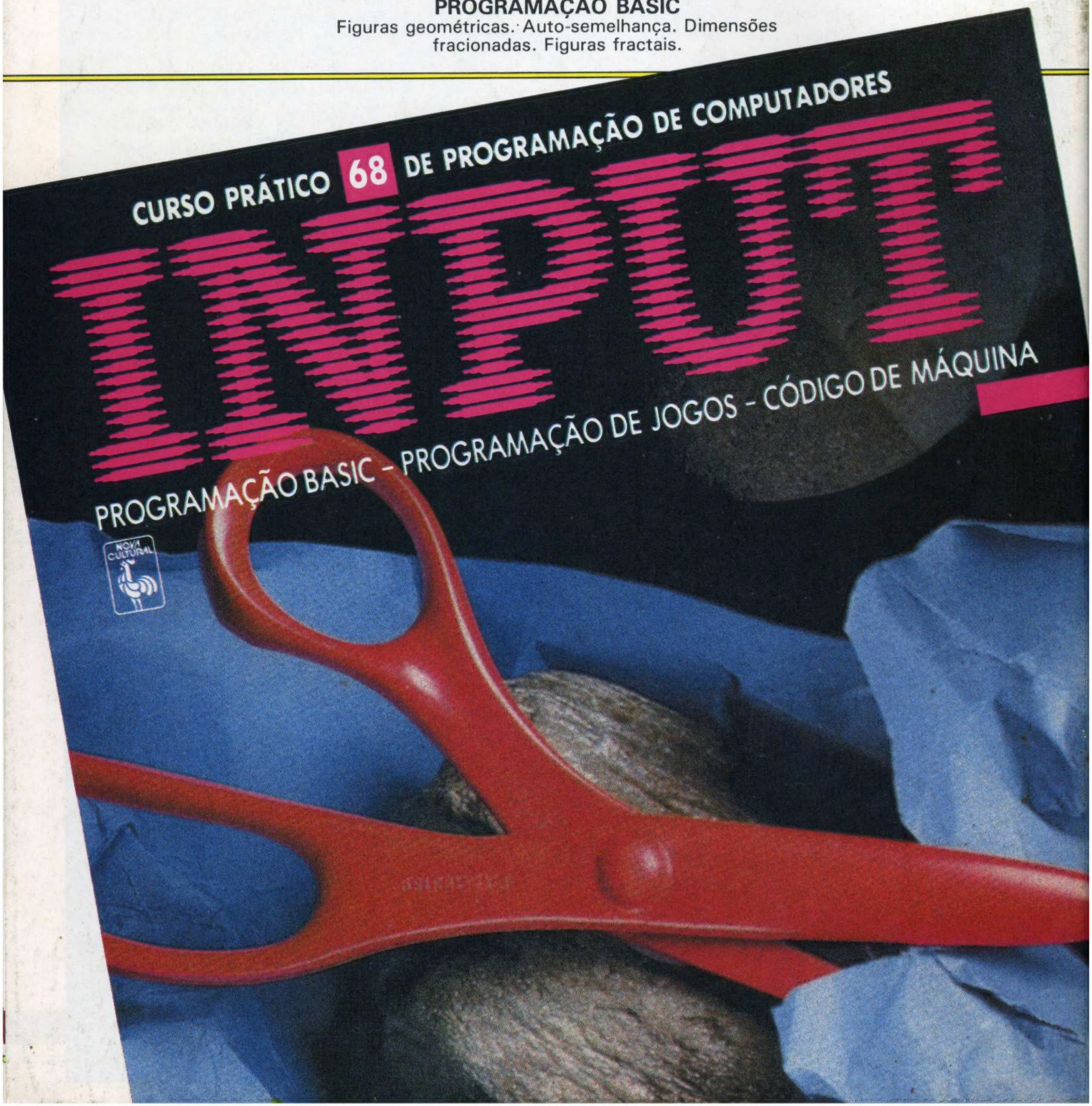
Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

NO PRÓXIMO NÚMERO

LINGUAGENS
Expressões matemáticas em LOGO. Variáveis e números. Nomes e sentenças. Comandos para listas.

PROGRAMAÇÃO BASIC
Novas funções matemáticas: maior múltiplo, menor múltiplo, resto de uma divisão, arredondamento, par ou ímpar.

PROGRAMAÇÃO BASIC
Como o computador enfrenta um blefe? Aleatoriedade. O uso da estatística. Alisamento exponencial. Soma cumulativa.

PROGRAMAÇÃO BASIC
Figuras geométricas. Auto-semelhança. Dimensões fracionadas. Figuras fractais.